# ldap3 Documentation

*Release 2.4*

**Giovanni Cannata**

**Sep 19, 2017**

# Contents

ldap3 is a pure Python LDAP 3 client library strictly conforming to RFC4510 and is released under the LGPL v3 open source license. RFC4510 is the current LDAP specification (June 2006) from IETF and obsoletes the previous LDAP RFCs 2251, 2830, 3771 (December 1997).

ldap3 can be used with any Python version starting from 2.6, including all Python 3 versions. It also works with PyPy and PyPy3.

**Warning:** ldap3 versioning follows SemVer. In version 2 the public API has slightly changed from version 1: some default values have been changed and the ldap3 namespace has been decluttered, removing redundant constants (look at the changelog for details). Also, the result code constants were moved to ldap3.core.results and the ldap3 custom exceptions were stored in ldap3.core.exceptions. If you experience errors in your existing code you should rearrange the import statements or explicitly set the defaults to their former values.

CHAPTER 1

# Contents

## The ldap3 project

ldap3 is a strictly RFC 4510 conforming LDAP v3 pure Python client library. The whole ldap3 library has been **written from scratch** and the **same codebase works with Python 2, Python 3, PyPy and PyPy3** on any system where it can gain access to the network via a Python interpreter and the Python Standard Library.

### License

The ldap3 library is open source software released under the **LGPL v3 license** (http://www.gnu.org/licenses/lgpl-3.0.html). This means that you can use the ldap3 library in any application (either open or proprietary). You can also copy, distribute and modify the ldap3 library provided that modifications are described and licensed for free under LGPL. Derivatives works of ldap3 can only be redistributed under LGPL, but applications that use the library don't have to be.

### RFCs Compliance

The ldap3 library strictly follows the latest (as of 2015) RFCs describing the LDAP v3 protocol:

- The **latest RFCs for LDAP v3** (RFCs 4510-4518, dated 2006) obsolete the previous RFCs specified in RFC3377 (2251-2256, 2829, 2830, 3371) for LDAP v3 and amend and clarify the LDAP protocol.
- All the ASN1 definitions are written from scratch to be current with RFC 4511.

To avoid unnecessary server and network load caused by poorly formed searches The ldap3 library deliberately doesn't follow the specification in RFC4511 (4.5.1.8.1) that states that in a Search operation "an empty list with no attributes requests the return of all user attributes.". Attributes must be explicitly requested or the ldap3.ALL_ATTRIBUTES must be used in the Search operation.

The library allows to send an empty member list while creating a GroupOfNames object, even if this is not allowed in the official LDAP v3 schema.

ldap3 allows communication over Unix sockets (ldapi:// scheme, LDAP over IPC) even if this is not required by any official LDAP RFCs.

## PEP8 Compliance

ldap3 is PEP8 compliant, except for line length. PEP8 (https://www.python.org/dev/peps/pep-0008/) is the standard coding style guide for the Python Standard Library and for many other Python projects. It provides a consistent way of writing code for maintainability and readability following the principle that "software is more read then written".

## Home Page

The home page of the ldap3 project is https://github.com/cannatag/ldap3

## Documentation

Documentation is available at http://ldap3.readthedocs.io. You can download a PDF copy of the manual at https://media.readthedocs.org/pdf/ldap3/stable/ldap3.pdf

## Download

The ldap3 package can be downloaded at https://pypi.python.org/pypi/ldap3. If you use a package manager that support the *wheel* format you can get the universal wheel package, and install it on any supported Python environment.

## Install

Install with **pip install ldap3**. If needed the library installs the `pyasn1` package. If you need Kerberos support you must install the `gssapi` package. ldap3 includes a backport (from Python 3.4.3) of ssl.check_hostnames to use on older (< 2.7.10) Python version. If you want to use a newer version of the check_hostnames feature you can install the `backports.ssl_check_hostnames` package that should be kept updated by its author with the latest Python release.

## GIT repository

You can download the latest released source code at https://github.com/cannatag/ldap3/tree/master

## Contributing to this project

ldap3 source is hosted on github. You can contribute to the ldap3 project on https://github.com/cannatag/ldap3 forking the project and submitting a *pull request* with your modifications.

## Continuous integration

Continuous integration for testing is at https://travis-ci.org/cannatag/ldap3

## Support

You can submit support tickets on https://github.com/cannatag/ldap3/issues/new

## Contact me

For information and suggestions you can contact me at cannatag@gmail.com. You can also open a support ticket on https://github.com/cannatag/ldap3/issues/new

## Donate

If you want to keep this project up and running you can send me an Amazon gift card. I will use it to improve my skills in the Information and Communication technology.

## Thanks to

- **Ilya Etingof**, the author of the *pyasn1* package for his excellent work and support.
- **Mark Lutz** for his *Learning Python* and *Programming Python* excellent books series and **John Goerzen** and **Brandon Rhodes** for their books *Foundations of Python Network Programming* (Second and Third edition). These books are wonderful tools for learning Python and this project owes a lot to them.
- **JetBrains** for donating to this project the Open Source license of *PyCharm 4 Professional*.
- **GitHub** for providing the *free source repository space and tools* used to develop this project.
- The **Python Software Foundation** for supporting the cloud lab infrastructure used for testing the library.

# ldap3 Tutorial

## Tutorial: Introduction to ldap3

---

**Note:** In this tutorial you will access a public demo of FreeIPA, available at https://ipa.demo1.freeipa.org (you must trust its certificate on first login). FreeIPA is a fully featured identity management solution, but for the purposes of this tutorial we're only interested in its LDAP server. Note that the demo server is periodically wiped, as described on the FreeIPA demo wiki page.

---

**Warning:** If you receive an `LDAPSocketReceiveError: error receiving data` exception the server could have closed the connection abruptly. You can easily reopen it with the `conn.bind()` method.

I assume that you already know what LDAP is, or at least have a rough idea of it. Even if you really don't know anything about LDAP, after reading this tutorial you should be able to access an LDAP compliant server and use it without bothering with the many glitches of the LDAP protocol.

### What LDAP is not

I'd rather want to be sure that you are aware of what LDAP **is not**:

- LDAP is not a server
- LDAP is not a database
- LDAP is not a network service

- LDAP is not an authentication procedure
- LDAP is not a user/password repository
- LDAP is not a specific open source neither a closed source product

It's important to know what LDAP is not because people usually call "LDAP" a peculiar part of what they use of the **Lightweight Directory Access Protocol**. LDAP is a *protocol* and as other 'trailing-P' words in the Internet ecosystem (HTTP, FTP, TCP, IP, ...) it is a set of rules you must follow to talk to an external server/database/service/procedure/repository/product (all the things in the above list). Data managed via LDAP are key/value(s) pairs grouped in a hierarchical structure. This structure is called the **DIT** (*Directory Information Tree*). LDAP doesn't specify how data is actually stored in the DIT neither how the user is authorized to access it. There are only a few data types that every LDAP server must recognize (some of them being very old and not used anymore). LDAP version 3 is also an extensible protocol, this means that a vendor can add features not in the LDAP specifications (using Controls and Extensions). Any LDAP server relies on a **schema** to know which data types, attributes and object it understands. A portion of the schema is standard (defined in the protocol itself), but each vendor can add attributes and object for specific purposes. The schema can also be extended (with administrative role) by the system administrator, the developer and the end user of the LDAP server. Keep in mind that "extending the schema" is something that is not defined in the LDAP protocol, so each vendor has developed different methods to add objects and attributes.

Being a protocol, LDAP is not related to any specific product and it is described in a set of **RFCs** (*Request for comments*, the official rules of the Internet ecosystem). Latest version of this rules is **version 3** documented in the RFC4510 (and subsequents RFCs) released in June 2006.

### A very brief history of LDAP

You may wonder why the "lightweight" in LDAP. Its ancestor, called **DAP** (*Directory Access Protocol*), was developed in the 1980s by the CCITT (now ITU-T), the *International Committee for Telephone and Telegraphy* (the venerable entity that gave us, among others, faxes and modem protocols we used in the pre-Internet era). DAP was a very heavy and hard-to-implement protocol (either for client and server components) and was not accessible via TCP/IP and its intended use was to standardize access to directory services (i.e. phone directories). In 1993 a simpler access protocol was invented at the University of Michigan to act as a gateway to the DAP world. Afterwards vendors developed server products that could understand LDAP directly and the gateway to DAP was soon removed. LDAP v3 was first documented in 1997 and its specifications was revised in 2006. These later specifications are strictly followed by the ldap3 library.

### Unicode everywhere

The LDAP protocol specifies that attribute names and their string values (usually in the Directory String LDAP syntax) must be stored in Unicode version 3.2 with the UTF-8 byte encoding. There are some limitations in the attribute names that can use only ASCII letters (upper and lowercase), numbers and the hypen character (but not as a leading character). Unicode is a standard to describe thousands of printed (even if not visible) characters but what goes over the wire when you interact with an LDAP server is only old plain bytes (with values ranging from 0 to 255 as usual), so the UTF-8 encoding is needed when talking to an LDAP server to convert the Unicode character to a valid byte (or multi-byte) representation. For this reason when you want to use a string value in any LDAP operation you must convert it to UTF-8 encoding. Your environment could have (and probably has) a different default encoding so the ldap3 library will try to convert from your default encoding to UTF-8 for you, but you may set a different input encoding with the `set_config_parameter('DEFAULT_ENCODING', 'my_encoding')` function of the ldap3 library. Values returned by the LDAP search operation are always encoded in UTF-8. This doesn't apply to other binary format, as Octect String that can use a different format.

### The ldap3 package

ldap3 is a fully compliant LDAP v3 client library following the official RFCs released in June 2006. It's written from scratch to be compatible with Python 2 and Python 3 and can be used on any machine where the Python interpreter can gain access to the network via the Python standard library.

Chances are that you find the ldap3 package already installed (or installable with your local package manager) on your machine, just try to **import ldap3** from your Python console. If you get an `ImportError` you need to install the package from PyPI via pip in the standard way:

```
pip install ldap3
```

**Warning:** If pip complains about certificates you should specify the path to the PyPI CA certificate with the –cert parameter:

```
pip install ldap3 --cert /path/to/the/DigiCert_High_Assurance_EV_Root_CA.pem
```

You can also download the source code from https://github.com/cannatag/ldap3 and install it with:

```
python setup.py install
```

ldap3 needs the **pyasn1** package (and will install it if not already present). This package is used to communicate with the server over the network. By default ldap3 uses the pyasn1 package only when sending data to the server. Data received from the server are decoded with an internal decoder, much faster (10x ) than the pyasn1 decoder.

### Accessing an LDAP server

ldap3 usage is straightforward: you define a Server object and a Connection object. Then you issue commands to the connection. A server can have any number of active connections with the same or a different *communication strategy*.

All the importable objects are available in the ldap3 namespace. At least you need to import the Server and the Connection object, and any additional constant you will use in your LDAP conversation (constants are defined in upper case):

```
>>> from ldap3 import Server, Connection, ALL
```

ldap3 specific exceptions are defined in the `ldap3.core.exceptions` package.

**Warning:** **A more pythonic LDAP**: LDAP operations look clumsy and hard-to-use because they reflect the old-age idea that time-consuming operations should be done on the client to not clutter and hog the server with unneeded elaboration. ldap3 includes a fully functional **Abstraction Layer** that lets you interact with the DIT in a modern and *pythonic* way. With the Abstraction Layer you don't need to directly issue any LDAP operation at all.

In the LDAP protocol the login operation is called **Bind**. A bind can be performed in 3 different ways: Anonymous Bind, Simple Password Bind, and SASL (*Simple Authentication and Security Layer*, allowing a larger set of authentication methods) Bind. You can think of the Anonymous Bind as of a *public* access to the LDAP server where no credentials are provided and the server applies some *default* access rules. With the Simple Password Bind and the SASL Bind you provide credentials that the LDAP server uses to determine your authorization level. Again, keep in mind that the LDAP standard doesn't define specific access rules and that the authorization mechanism is not specified at all. So each LDAP server vendor can have a different method for authorizing the user to access data stored in the DIT.

ldap3 lets you choose the method that the client will use to connect to the server with the `client_strategy` parameter of the Connection object. There are four strategies that can be used for establishing a connection: SYNC, ASYNC, RESTARTABLE and REUSABLE. As a general rule, in synchronous strategies (**SYNC**, **RESTARTABLE**) all LDAP operations return a boolean: `True` if they're successful, `False` if they fail; in asynchronous strategies (**ASYNC**, **REUSABLE**) all LDAP operations (except Bind that always returns a boolean) return a number, the *message_id* of the request. With asynchronous strategies you can send multiple requests without waiting for responses and then you get each response with the `get_response(message_id)` method of the Connection object as you need it. ldap3 will raise an exception if the response has not yet arrived after a specified time. In the `get_response()` method this timeout value can be set with the `timeout` parameter to the number of seconds to wait for the response to appear (default is 10 seconds). If you use the `get_request=True` in the `get_response()` parameter you get the request dictionary back.

Asynchronous strategies are thread-safe and are useful with slow servers or when you have many requests with the same Connection object in multiple threads. Usually you will use synchronous strategies only.

The **LDIF** strategy is used to create a stream of LDIF-CHANGEs. (LDIF stands for *LDAP Data Interchange Format*, textual standard used to describe the changes performed by LDAP operations). The MOCK_SYNC strategy can be used to emulate a fake LDAP server to test your application without the need of a real LDAP server.

---

**Note:** In this tutorial you will use the default SYNC communication strategy. If you keep loosing connection to the server you can use the RESTARTABLE communication strategy that tries to reconnect and resend the operation when the link to the server fails.

---

Let's start accessing the server with an anonymous bind:

```
>>> server = Server('ipa.demo1.freeipa.org')
>>> conn = Connection(server)
>>> conn.bind()
True
```

or shorter:

```
>>> conn = Connection('ipa.demo1.freeipa.org', auto_bind=True)
```

Hardly it could be simpler than that. The `auto_bind=True` parameter forces the Bind operation while creating the Connection object. You have now a full working anonymous session open and bound to the server with a *synchronous* communication strategy:

```
>>> print(conn)
ldap://ipa.demo1.freeipa.org:389 - cleartext - user: None - bound - open - <local:␣
↪192.168.1.101:49813 - remote: 209.132.178.99:389> -
tls not started - listening - SyncStrategy - internal decoder
```

With `print(conn)` you ask the connection for its status and get back a lot of information:

| | |
|---|---|
| ldap://ipa.demo1.freeipa.org:389 | the server URL (scheme, name and port we are connected to) |
| cleartext | the kind of connection the server is listening to |
| user: None | the credentials used, in this case None means an anonymous binding |
| bound | the status of the LDAP session |
| open | the status of the underlying TCP/IP session |
| <local: 192.168.1.101:51038 - remote: 23.20.46.132:389> | the local and remote communication endpoints |
| tls not started | the status of the TLS (Transport Layer Security) session |
| listening | the status of the communication strategy |
| SyncStrategy | the communication strategy used |
| internal decoder | which BER decoder the connection is using (pyasn1 or the faster internal decoder) |

**Note:** Object representation: the ldap3 library uses the following object representation rule: when you use `str()` you get back information about the status of the object in a human readable format, when you use `repr()` you get back a string you can use in the Python console to recreate the object. `print` always return the `str()` representation. Typing at the >>> prompt always return the `repr` representation.

If you ask for the `repr()` representation of the conn object you can get a string to recreate the object:

```
>>> conn
Connection(server=Server(host='ipa.demo1.freeipa.org', port=389, use_ssl=False, get_
↪info='NO_INFO'), auto_bind='NONE',
version=3, authentication='ANONYMOUS', client_strategy='SYNC', auto_referrals=True,␣
↪check_names=True, read_only=False,
lazy=False, raise_exceptions=False, fast_decoder=True)
```

If you just copy and paste the object representation at the >>> prompt you can instantiate a new object similar to the original one. This is helpful when experimenting in the interactive console and works for most of the ldap3 library objects:

```
>>> server
Server(host='ipa.demo1.freeipa.org', port=389, use_ssl=False, get_info='NO_INFO')
```

**Note:** The tutorial is intended to be used from the *REPL* (Read, Evaluate, Print, Loop), the interactive Python command line where you can directly type Python statements at the **>>>** prompt. The REPL implicitly use the `repr()` representation for showing the output of a statement. If you instead want the `str()` representation you must explicitly use the `print()` statement.

### Getting information from the server

The LDAP protocol specifies that an LDAP server must return some information about itself. You can request them with the `get_info=ALL` parameter and access them with the `.info` attribute of the Server object:

```
>>> server = Server('ipa.demo1.freeipa.org', get_info=ALL)
>>> conn = Connection(server, auto_bind=True)
>>> server.info
DSA info (from DSE):
  Supported LDAP Versions: 2, 3
  Naming Contexts:
    cn=changelog
```

```
    dc=demo1,dc=freeipa,dc=org
    o=ipaca
  Alternative Servers: None
  Supported Controls:
    1.2.840.113556.1.4.319 - LDAP Simple Paged Results - Control - RFC2696
    1.2.840.113556.1.4.473 - Sort Request - Control - RFC2891
    1.3.6.1.1.13.1 - LDAP Pre-read - Control - RFC4527
    1.3.6.1.1.13.2 - LDAP Post-read - Control - RFC4527
    1.3.6.1.4.1.1466.29539.12 - Chaining loop detect - Control - SUN microsystems
    1.3.6.1.4.1.42.2.27.8.5.1 - Password policy - Control - IETF DRAFT behera-ldap-
↪password-policy
    1.3.6.1.4.1.42.2.27.9.5.2 - Get effective rights - Control - IETF DRAFT draft-
↪ietf-ldapext-acl-model
    1.3.6.1.4.1.42.2.27.9.5.8 - Account usability - Control - SUN microsystems
    1.3.6.1.4.1.4203.1.9.1.1 - LDAP content synchronization - Control - RFC4533
    1.3.6.1.4.1.4203.666.5.16 - LDAP Dereference - Control - IETF DRAFT draft-
↪masarati-ldap-deref
    2.16.840.1.113730.3.4.12 - Proxied Authorization (old) - Control - Netscape
    2.16.840.1.113730.3.4.13 - iPlanet Directory Server Replication Update␣
↪Information - Control - Netscape
    2.16.840.1.113730.3.4.14 - Search on specific database - Control - Netscape
    2.16.840.1.113730.3.4.15 - Authorization Identity Response Control - Control -␣
↪RFC3829
    2.16.840.1.113730.3.4.16 - Authorization Identity Request Control - Control -␣
↪RFC3829
    2.16.840.1.113730.3.4.17 - Real attribute only request - Control - Netscape
    2.16.840.1.113730.3.4.18 - Proxy Authorization Control - Control - RFC6171
    2.16.840.1.113730.3.4.19 - Chaining loop detection - Control - Netscape
    2.16.840.1.113730.3.4.2 - ManageDsaIT - Control - RFC3296
    2.16.840.1.113730.3.4.20 - Mapping Tree Node - Use one backend [extended] -␣
↪Control - openLDAP
    2.16.840.1.113730.3.4.3 - Persistent Search - Control - IETF
    2.16.840.1.113730.3.4.4 - Netscape Password Expired - Control - Netscape
    2.16.840.1.113730.3.4.5 - Netscape Password Expiring - Control - Netscape
    2.16.840.1.113730.3.4.9 - Virtual List View Request - Control - IETF
    2.16.840.1.113730.3.8.10.6 - OTP Sync Request - Control - freeIPA
  Supported Extensions:
    1.3.6.1.4.1.1466.20037 - StartTLS - Extension - RFC4511-RFC4513
    1.3.6.1.4.1.4203.1.11.1 - Modify Password - Extension - RFC3062
    1.3.6.1.4.1.4203.1.11.3 - Who am I - Extension - RFC4532
    2.16.840.1.113730.3.5.10 - Distributed Numeric Assignment Extended Request -␣
↪Extension - Netscape
    2.16.840.1.113730.3.5.12 - Start replication request - Extension - Netscape
    2.16.840.1.113730.3.5.3 - Transaction Response Extended Operation - Extension -␣
↪Netscape
    2.16.840.1.113730.3.5.4 - iPlanet Replication Response Extended Operation -␣
↪Extension - Netscape
    2.16.840.1.113730.3.5.5 - iPlanet End Replication Request Extended Operation -␣
↪Extension - Netscape
    2.16.840.1.113730.3.5.6 - iPlanet Replication Entry Request Extended Operation -␣
↪Extension - Netscape
    2.16.840.1.113730.3.5.7 - iPlanet Bulk Import Start Extended Operation -␣
↪Extension - Netscape
    2.16.840.1.113730.3.5.8 - iPlanet Bulk Import Finished Extended Operation -␣
↪Extension - Netscape
    2.16.840.1.113730.3.5.9 - iPlanet Digest Authentication Calculation Extended␣
↪Operation - Extension - Netscape
    2.16.840.1.113730.3.6.5 - Replication CleanAllRUV - Extension - Netscape
```

```
    2.16.840.1.113730.3.6.6 - Replication Abort CleanAllRUV - Extension - Netscape
    2.16.840.1.113730.3.6.7 - Replication CleanAllRUV Retrieve MaxCSN - Extension -␣
↪Netscape
    2.16.840.1.113730.3.6.8 - Replication CleanAllRUV Check Status - Extension -␣
↪Netscape
    2.16.840.1.113730.3.8.10.1 - KeyTab set - Extension - FreeIPA
    2.16.840.1.113730.3.8.10.3 - Enrollment join - Extension - FreeIPA
    2.16.840.1.113730.3.8.10.5 - KeyTab get - Extension - FreeIPA
  Supported SASL Mechanisms:
    EXTERNAL, GSS-SPNEGO, GSSAPI, DIGEST-MD5, CRAM-MD5, PLAIN, LOGIN, ANONYMOUS
  Schema Entry:
    cn=schema
Vendor name: 389 Project
Vendor version: 389-Directory/1.3.3.8 B2015.036.047
Other:
  dataversion:
    020150912040104020150912040104020150912040104
  changeLog:
    cn=changelog
  lastchangenumber:
    3033
  firstchangenumber:
    1713
  lastusn:
    8284
  defaultnamingcontext:
    dc=demo1,dc=freeipa,dc=org
  netscapemdsuffix:
    cn=ldap://dc=ipa,dc=demo1,dc=freeipa,dc=org:389
  objectClass:
    top
```

This server (like most LDAP servers) lets an anonymous user to know a lot about it:

| | | |
|---|---|---|
| Supported LDAP Versions | 2, 3 | Server supports LDAP 2 and 3 |
| Naming contexts | <...> | Server stores information for 3 different DIT partitions |
| Alternative servers | None | This is the only replica of the database |
| Supported Controls | <...> | Optional controls that can be sent in a request operation |
| Supported Extentions | <...> | Additional extended operations understood by the server |
| Supported SASL Mechanisms | <...> | Different additional SASL authentication mechanisms available |
| Schema Entry | cn=schema | The location of the schema in the DIT |
| Vendor name | 389 Project | The brand/mark/name of this LDAP server |
| Vendor version | 389-Directory/1.3.3 ... | The version of this LDAP server |
| Other | ... | Additional information provided by the server |

From this response we know that this server is a stand-alone LDAP server that can hold entries in the dc=demo1,dc=freeipa,dc=org context, that supports various SASL access mechanisms and that is based on the 389 Directory Service server. Furthermore in the Supported Controls we can see it supports "paged searches", and the "who am i" and "StartTLS" extended operations in Supported Extensions.

---

**Note:** Controls vs Extensions: in LDAP a *Control* is some additional information that can be attached to any LDAP request or response, while an *Extension* is a custom request that can be sent to the LDAP server in an **Extended Operation** Request. A Control usually modifies the behaviour of a standard LDAP operation, while an Extension is a completely new kind of operation that each vendor decides to include in its LDAP server implementation. An LDAP

server declares which controls and which extendend operations it understands. The ldap3 library decodes the known supported controls and extended operation and includes a brief description and a reference to the relevant RFC in the `.info` attribute (when known). Not all controls or extensions are intended to be used by clients. Some controls and extensions are used by servers that hold a replica or a data partition. Unfortunately in the LDAP specifications there is no way to specify if such extensions are reserved for a server (**DSA**, *Directory Server Agent* in LDAP parlance) to server communication (for example in replicas or partitions management) or can be used by clients (**DUA**, *Directory User Agent*). Because the LDAP protocols doesn't provide a specific way for DSAs to communicate with each other, a DSA actually presents itself as a DUA to another DSA.

---

An LDAP server store information about known *types* in its **schema**. The schema includes all information needed by a client to correctly performs LDAP operations. Let's examine the LDAP server schema:

```
>>> server.schema
DSA Schema from: cn=schema
  Attribute types:{'ipaNTTrustForestTrustInfo': Attribute type: 2.16.840.1.113730.3.8.
↪11.17
  Short name: ipaNTTrustForestTrustInfo
  Description: Forest trust information for a trusted domain object
  Equality rule: octetStringMatch
  Syntax: 1.3.6.1.4.1.1466.115.121.1.40 [('1.3.6.1.4.1.1466.115.121.1.40', 'LDAP_
↪SYNTAX', 'Octet String', 'RFC4517')]
  'ntUserCreateNewAccount': Attribute type: 2.16.840.1.113730.3.1.42
  Short name: ntUserCreateNewAccount
  Description: Netscape defined attribute type
  Single Value: True
  Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
  Extensions:
    X-ORIGIN: Netscape NT Synchronization
  'passwordGraceUserTime': Attribute type: 2.16.840.1.113730.3.1.998
  Short name: passwordGraceUserTime, pwdGraceUserTime
  Description: Netscape defined password policy attribute type
  Single Value: True
  Usage: Directory operation
  Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
  Extensions:
    X-ORIGIN: Netscape Directory Server
  'nsslapd-ldapilisten': Attribute type: 2.16.840.1.113730.3.1.2229
  Short name: nsslapd-ldapilisten
  Description: Netscape defined attribute type
  Single Value: True
  Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
  Extensions:
    X-ORIGIN: Netscape Directory Server
  'bootParameter': Attribute type: 1.3.6.1.1.1.1.23
  Short name: bootParameter
  Description: Standard LDAP attribute type
  Syntax: 1.3.6.1.4.1.1466.115.121.1.26 [('1.3.6.1.4.1.1466.115.121.1.26', 'LDAP_
↪SYNTAX', 'IA5 String', 'RFC4517')]
  Extensions:
    X-ORIGIN: RFC 2307

  <...long list of descriptors...>
```

The schema is a very long list that describes what kind of data types the LDAP server understands. It also specifies

what attributes can be stored in each class. Some classes are containers for other entries (either container or leaf) and are used to build the hierarchy of the DIT. Container entries can have attributes too. One important specification in the schema is if the attribute is *multi-valued* or not. A multi-valued attribute can store one or more values. Every LDAP server must at least support the standard LDAP3 schema but can have additional custom classes and attributes. The schema defines also the *syntaxes* and the *matching rules* of the different kind of data types stored in the LDAP.

---

**Note:** Object classes and attributes are independent objects. An attribute is not a "child" of a class neither a class is a "parent" of any attribute. Classes and attributes are linked in the schema with the `MAY` and `MUST` options of the object class definition that specify what attributes an entry can contain and which of them are mandatory.

---

---

**Note:** There are 3 different types of object classes: **ABSTRACT** (used only when defining the class hierarchy), **STRUCTURAL** (used to create concrete entries) and **AUXILIARY** (used to add additional attributes to an entry). Only one structural class can be used in an entry, while many auxiliary classes can be added to the same entry. Adding an object class to an entry simply means that the attributes defined in that object class can be stored in that entry.

---

If the ldap3 library is aware of the schema used by the LDAP server it will try to automatically convert data retrieved by the Search operation to their representation. An integer will be returned as an int, a generalizedDate as a datetime object and so on. If you don't read the schema all the values are returned as bytes and unicode strings. You can control this behaviour with the `get_info` parameter of the Server object and the `check_names` parameter of the Connection object.

### Logging into the server

You haven't provided any credentials to the server yet, but you received a response anyway. This means that LDAP allows users to perform operations anonymously without declaring their identity. Obviously what the server returns to an anonymous connection is someway limited. This makes sense because originally the DAP protocol was intended for reading phone directories, as in a printed book, so its content could be read by anyone.

If you want to establish an authenticated session you have two options: Simple Password and SASL. With Simple Password you provide a **DN** (*Distinguished Name*) and a password. The server checks if your credentials are valid and permits or denies access to the elements of the DIT. SASL provides additional methods to identify the user, as an external certificate or a Kerberos ticket.

---

**Note:** Distinguished Names: the DIT is a hierarchical structure, as a filesystem. To identify an entry you must specify its *path* in the DIT starting from the leaf that represents the entry up to the top of the Tree. This path is called the **Distinguished Name** (DN) of an entry and is constructed with key-value pairs, separated by a comma, of the names of the entries that form the path from the leaf up to the top of the Tree. The DN of an entry is unique throughout the DIT and changes only if the entry is moved into another container within the DIT. The parts of the DN are called **Relative Distinguished Name** (RDN) because are unique only in the context where they are defined. So, for example, if you have a *inetOrgPerson* entry with RDN `cn=Fred` that is stored in an *organizational unit* with RDN `ou=users` that is stored in an *organization* with RDN `o=company` the DN of the *inetOrgPerson* entry will be `cn=Fred,ou=users,o=company`. The RDN value must be unique in the context where the entry is stored, but there is no specification in the LDAP schema on which attribute to use as RDN for a specific class. LDAP also support a (quite obscure) "multi-rdn" naming option where each part of the RDN is separated with the + character, as in `cn=Fred+sn=Smith`.

---

**Warning:** Accessing Active Directory: with ldap3 you can also connect to an Active Directory server with the NTLM v2 protocol:

```
>>> from ldap3 import Server, Connection, ALL, NTLM
>>> server = Server('servername', get_info=ALL)
>>> conn = Connection(server, user="Domain\\User", password="password",␣
↪authentication=NTLM)
```

This kind of authentication is not part of the LDAP 3 RFCs but uses a proprietary Microsoft authentication mechanism named SICILY. ldap3 implements it because it's much easier to use this method than Kerberos to access Active Directory.

Now try to ask to the server who you are:

```
>>> conn.extend.standard.who_am_i()
```

We have used and Extended Operation, conveniently packaged in a function of the `ldap3.extend.standard` package, and get an empty response. This means you have no authentication status on the server, so you are an **anonymous** user. This doesn't mean that you are unknown to the server, actually you have a session open with it, so you can send additional operation requests. Even if you don't send the anonymous bind operation the server will accept any operation requests as an anonymous user, establishing a new session if needed.

---

**Note:** The `extend` namespace. The connection object has a special namespace called "extend" where more complex operations are defined. This namespace include a `standard` section and a number of specific vendor sections. In these sections you can find methods to perform tricky or hard-to-implement operations. For example in the `microsoft` section you can find a method to easily change the user password, and in the `novell` section a method to apply transaction to groups of LDAP operations. In the `standard` section you can also find an easy way to perform a paged search via generators.

---

**Warning:** Opening vs Binding: the LDAP protocol provides a Bind and an Unbind operation but, for historical reasons, they are not symmetric. As any TCP connection the communication socket must be *open* before binding to the server . This is implicitly done by the ldap3 package when you issue a `bind()` or another operation or can be esplicity done with the `open()` method of the Connection object. The Unbind operation is actually used to *terminate* the connection, both ending the session and closing the socket. After the `unbind()` operation the connection cannot be used anymore. If you want to access as another user or change the current session to an anonymous one, you must issue `bind()` again. The ldap3 library allows you to use the `rebind()` method to access the same connection as a different user. You must use `unbind()` only when you want to close the network socket.

Try to specify a valid user:

```
>>> conn = Connection(server, 'uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,
↪dc=org', 'Secret123', auto_bind=True)
>>> conn.extend.standard.who_am_i()
'dn: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org'
```

Now the server knows that you are a recognized user and the `who_am_i()` extended operation returns your identity.

### Establishing a secure connection

If you check the connection info you can see that the Connection is using a cleartext (insecure) channel:

```
>>> print(conn)
ldap://ipa.demo1.freeipa.org:389 - **cleartext** - user: uid=admin,cn=users,
↪cn=accounts,dc=demo1,dc=freeipa,dc=org - bound - open - <local: 192.168.1.101:50164␣
↪- remote: 209.132.178.99:**389**> - **tls not started** - listening - SyncStrategy -
↪ internal decoder'
```

This means that credentials pass unencrypted over the wire, so they can be easily captured by network eavesdroppers (with unencrypted connections a network sniffer can be easily used to capture passwords and other sensitive data). The LDAP protocol provides two ways to secure a connection: **LDAP over TLS** (or over SSL) and the **StartTLS** extended operation. Both methods establish a secure TLS connection: the former secure with TLS the communication channel as soon as the connection is open, while the latter can be used at any time on an already open unsecure connection to secure it issuing the StartTLS operation.

**Warning:** LDAP URL scheme: a cleartext connection to a server can be expressed in the URL with the **ldap://** scheme, while LDAP over TLS can be indicated with **ldaps://** even if this is not specified in any of the LDAP RFCs. If a scheme is included in the server name while creating the Server object, the ldap3 library opens the proper port, unencrypted or with the specified TLS options (or the default TLS options if none is specified).

---

**Note:** Default port numbers: the default port for cleartext (unsecure) communication is **389**, while the default for LDAP over TLS (secure) communication is **636**. Note that because you can start a session on the 389 port and then raise the security level with the StartTLS operation, you can have a secure communication even on the 389 port (usually considered unsecure). Obviously the server can listen on additional or different ports. When defining the Server object you can specify which port to use with the `port` parameter. Keep this in mind if you need to connect to a server behind a firewall.

---

Now try to use the StartTLS extended operation:

```
>>> conn.start_tls()
True
```

if you check the connection status you can see that the session is on a secure channel now, even if started on a cleartext connection:

```
>>> print(conn)
ldap://ipa.demo1.freeipa.org:389 - cleartext - user: uid=admin,cn=users,cn=accounts,
↪dc=demo1,dc=freeipa,dc=org - bound - open - <local: 192.168.1.101:50910 - remote:␣
↪209.132.178.99:389> - tls started - listening - SyncStrategy - internal decoder
```

To start the connection on a SSL socket:

```
>>> server = Server('ipa.demo1.freeipa.org', use_ssl=True, get_info=ALL)
>>> conn = Connection(server, 'uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,
↪dc=org', 'Secret123', auto_bind=True)
>>> print(conn)
ldaps://ipa.demo1.freeipa.org:636 - ssl - user: uid=admin,cn=users,cn=accounts,
↪dc=demo1,dc=freeipa,dc=org - bound - open - <local: 192.168.1.101:51438 - remote:␣
↪209.132.178.99:636> - tls not started - listening - SyncStrategy - internal decoder
```

Either with the former or the latter method the connection is now encrypted. We haven't specified any TLS option, so there is no checking of certificate validity. You can customize the TLS behaviour providing a Tls object to the Server object using the security context configuration:

```
>>> from ldap3 import Tls
>>> import ssl
>>> tls_configuration = Tls(validate=ssl.CERT_REQUIRED, version=ssl.PROTOCOL_TLSv1)
>>> server = Server('ipa.demo1.freeipa.org', use_ssl=True, tls=tls_configuration)
>>> conn = Connection(server)
>>> conn.open()
...
ldap3.core.exceptions.LDAPSocketOpenError: (LDAPSocketOpenError('socket ssl wrapping␣
↪error: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:600)',),)
```

In this case, using the FreeIPA demo server we get a LDAPSocketOpenError exception because the certificate cannot be verified. You can configure the Tls object with a number of options. Look at SSL and TLS for more information.

The FreeIPA server doesn't return a valid certificate so to continue the tutorial let's revert the certificate validation to CERT_NONE:

```
>>> tls_configuration.validate = ssl.CERT_NONE
```

### Connection context manager

The Connection object responds to the context manager protocol, so you can perform LDAP operations with automatic open, bind and unbind as in the following example:

```
>>> with Connection(server, 'uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
↪', 'Secret123') as conn:
        conn.search('dc=demo1,dc=freeipa,dc=org', '(&(objectclass=person)(uid=admin))
↪', attributes=['sn','krbLastPwdChange', 'objectclass'])
        entry = conn.entries[0]
True
>>> conn.bound
False
>>> entry
DN: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
krbLastPwdChange: 2016-10-09 10:01:18+00:00
objectclass: top
             person
             posixaccount
             krbprincipalaux
             krbticketpolicyaux
             inetuser
             ipaobject
             ipasshuser
             ipaSshGroupOfPubKeys
sn: Administrator
```

When the Connection object exits the context manager it retains the state it had before entering the context. The connection is always open and bound while in context. If the connection was not bound to the server when entering the context the Unbind operation will be tried when you leave the context even if the operations in the context raise an exception.

## Tutorial: searching LDAP entries

**Warning:** **A more pythonic LDAP**: LDAP operations look clumsy and hard-to-use because they reflect the old-age idea that time-consuming operations should be done on the client to not clutter and hog the server with unneeded elaboration. ldap3 includes a fully functional **Abstraction Layer** that lets you interact with the DIT in a modern and *pythonic* way. With the Abstraction Layer you don't need to directly issue any LDAP operation at all.

### Finding entries

To find entries in the DIT you must use the **Search** operation. This operation has a number of parameters, but only two of them are mandatory:

- `search_base`: the location in the DIT where the search will start
- `search_filter`: a string that describes what you are searching

Search filters are based on assertions and look odd when you're unfamiliar with their syntax. One *assertion* is a bracketed expression that affirms something about an attribute and its values, as `(givenName=John)` or `(maxRetries>=10)`. On the server each assertion resolves to True, False or Undefined (that is treated as False) for one or more entries in the DIT. Assertions can be grouped in boolean groups where all assertions (**and** group, specified with `&`) or at least one assertion (**or** group, specified with `|`) must be True. A single assertion can be negated (**not** group, specified with `!`). Each group must be bracketed, allowing for recursive filters. Operators allowed in an assertion are = (**equal**), <= (**less than or equal**), >= (**greater than or equal**), =* (**present**), ~= (**approximate**) and := (**extensible**). Surprisingly the *less than* and the *greater than* operators don't exist in the LDAP filter syntax. The *aproximate* and the *extensible* are someway obscure and seldom used. In an equality filter you can use the * character as a wildcard.

For example, to search for all users named John with an email ending with '@example.org' the filter will be `(&(givenName=John)(mail=*@example.org))`, to search for all users named John or Fred with an email ending in '@example.org' the filter will be `(&(|(givenName=Fred)(givenName=John))(mail=*@example.org))`, while to search for all users that have a givenName different from Smith the filter will be `(!(givenName=Smith))`.

Long search filters can easily become hard to understand so it may be useful to divide the text on multiple indented lines:

```
(&
    (|
        (givenName=Fred)
        (givenName=John)
    )
    (mail=*@example.org)
)
```

Let's search all users in the FreeIPA demo LDAP server:

```
>>> from ldap3 import Server, Connection, ALL
>>> server = Server('ipa.demo1.freeipa.org', get_info=ALL)
>>> conn = Connection(server, 'uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,
↪dc=org', 'Secret123', auto_bind=True)
>>> conn.search('dc=demo1,dc=freeipa,dc=org', '(objectclass=person)')
True
>>> conn.entries
[DN: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
, DN: uid=manager,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
, DN: uid=employee,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
```

```
, DN: uid=helpdesk,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
]
```

Here you request all the entries of class *person*, starting from the *dc=demo1,dc=freeipa,dc=org* context with the default subtree scope. You have not requested any attribute, so in the response we get only the Distinguished Name of the found entries.

---

**Note:** response vs result: in ldap3 every operation has a *result* that is stored in the `result` attribute of the Connection in sync strategies. Search operations store the found entries in the `response` attribute of the Connection object. For async strategies you must use the `get_response(id)` method that returns a tuple in the form of (response, result). If you use the `get_request=True` parameter you ask `get_response()` to return the request dictionary too so te returned tuple will be (response, result, request).

---

Now let's try to request some attributes from the admin user:

```
>>> conn.search('dc=demo1,dc=freeipa,dc=org', '(&(objectclass=person)(uid=admin))',␣
↪attributes=['sn', 'krbLastPwdChange', 'objectclass'])
True
>>> conn.entries[0]
DN: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS: Read - READ␣
↪TIME: 2016-10-09T20:39:32.711000
krbLastPwdChange: 2016-10-09 10:01:18+00:00
objectclass: top
             person
             posixaccount
             krbprincipalaux
             krbticketpolicyaux
             inetuser
             ipaobject
             ipasshuser
             ipaSshGroupOfPubKeys
             ipaNTUserAttrs
sn: Administrator
```

**Warning:** When using attributes in a search filter it's a good habit to always request for the *structural class* of the objects you expect to retrieve. You cannot be sure that the attribute you're serching for is not used is some other object class, and even if you are sure that no other object class uses it this could always change in the future when someone extends the schema with an object class that uses that very same attribute, and your program suddenly breaks with no apparent reason.

Note that the `entries` attribute of the Connection object is derived from the ldap3 *Abstraction Layer* and it's specially crafted to be used in interactive mode at the >>> prompt. It gives a visual representation of the entry data structure where each value is, according to the schema, properly formatted (the date value in krbLastPwdChange is actually stored as `b'20161009010118Z'`, but it's shown as a Python date object). Attributes can be queried either as a class or as a dict, with some additional features as case-insensitivity and blank-insensitivity. You can get the formatted value and the raw value (the value actually returned by the server) in the `values` and `raw_values` attributes:

```
>>> entry = conn.entries[0]
>>> entry.krbLastPwdChange
krbLastPwdChange: 2016-10-09 10:01:18+00:00
>>> entry.KRBLastPwdCHANGE
krbLastPwdChange: 2016-10-09 10:01:18+00:00
>>> entry['krbLastPwdChange']
```

```
krbLastPwdChange: 2016-10-09 10:01:18+00:00
>>> entry['KRB LAST PWD CHANGE']
krbLastPwdChange 2016-10-09 10:01:18+00:00

>>> entry.krbLastPwdChange.values
[datetime.datetime(2016, 10, 9, 10, 1, 18, tzinfo=OffsetTzInfo(offset=0, name='UTC'))]
>>> entry.krbLastPwdChange.raw_values
[b'20161009010118Z']
```

Note that the entry status is *Read*. This is not relevant if you only need to retrieve the entries from the DIT but it's vital if you want to take advantage of the ldap3 Abstraction Layer making it *Writable* and change or delete its content via the Abstraction Layer. The Abstraction Layer also records the time of the last data read operation for the entry.

In the previous search operations you specified `dc=demo1,dc=freeipa,dc=org` as the base of our search, but the entries we got back were in the `cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org` context of the DIT. So the server has, with no apparent reason, walked down every context under the base applying the filter to each of the entries in the sub-containers. The server actually performed a *whole subtree* search. Other possible kinds of searches are the *single level* search (that searches only in the level specified in the base) and the *base object* search (that searches only in the attributes of the entry specified in the base). What changes in this different kinds of search is the 'breath' of the portion of the DIT that is searched. This breath is called the **scope** of the search and can be specified with the `search_scope` parameter of the search operation. It can take three different values: `BASE`, `LEVEL` and `SUBTREE`. The latter value is the default for the search opertion, so this clarifies why you got back all the entries in the sub-containers of the base in previous searches.

You can have a LDIF representation of the response of a search with:

```
>>> print(conn.entries[0].entry_to_ldif())
version: 1
dn: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org
objectclass: top
objectclass: person
objectclass: posixaccount
objectclass: krbprincipalaux
objectclass: krbticketpolicyaux
objectclass: inetuser
objectclass: ipaobject
objectclass: ipasshuser
objectclass: ipaSshGroupOfPubKeys
krbLastPwdChange: 20161009010118Z
sn: Administrator
# total number of entries: 1
```

---

**Note:** LDIF stands for *LDAP Data Interchange Format* and is a textual standard used to describe two different aspects of LDAP: the content of an entry (**LDIF-CONTENT**) and the changes performed on an entry with an LDAP operation (**LDIF-CHANGE**). LDIF-CONTENT is used to describe LDAP entries in an stream (i.e. a file or a socket), while LDIF-CHANGE is used to describe the Add, Delete, Modify and ModifyDn operations.

*These two formats have different purposes and cannot be mixed in the same stream.*

---

or you can save the response to a JSON string:

```
>>> print(entry.entry_to_json())
{
    "attributes": {
        "krbLastPwdChange": [
            "2016-10-09 10:01:18+00:00"
```

```
        ],
        "objectclass": [
            "top",
            "person",
            "posixaccount",
            "krbprincipalaux",
            "krbticketpolicyaux",
            "inetuser",
            "ipaobject",
            "ipasshuser",
            "ipaSshGroupOfPubKeys"
        ],
        "sn": [
            "Administrator"
        ]
    },
    "dn": "uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org"
```

### Searching for binary values

To search for a binary value you must use the RFC4515 ASCII escape sequence for each unicode point in the search assertion. ldap3 provides the helper function *escape_bytes(byte_value)* in ldap3.utils.conv to properly escape a byte sequence:

```
>>> from ldap3.utils.conv import escape_bytes
>>> unique_id = b'\xca@\xf2k\x1d\x86\xcaL\xb7\xa2\xca@\xf2k\x1d\x86'
>>> search_filter = '(nsUniqueID=' + escape_bytes(unique_id) + ')'
>>> conn.search('dc=demo1,dc=freeipa,dc=org', search_filter, attributes=['nsUniqueId
↪'])
```

`search_filter` will contain `(nsUniqueID=\\ca\\40\\f2\\6b\\1d\\86\\ca\\4c\\b7\\a2\\ca\\40\\f2\\6b\`
The \xx escaping format is specific to the LDAP protocol.

### Entries Retrieval

Raw values for the attributes retrieved in an entry are stored in the `raw_attributes` dictonary in the `response` attribute.

ldap3 provides some **standard formatters** used to format the values retrieved in a Search operation as specified by the RFCs according to the current schema syntaxes. If the schema is known (with `get_info=SCHEMA` or `get_info=ALL` in the Server object) and the `check_names` parameter of the Connection object is set to True, the `attributes` attribute is populated with the formatted values. If the attribute is defined in the schema as *multi valued* then the attribute value is returned as a list (even if only a single value is present) else it's returned as a single value.

**Custom formatters** can be added to specify how attribute values are returned. A formatter must be a callable that receives a bytes value and returns an object.

### What about empty attributes?

In LDAP an attribute must always have a value. An attribute with no value is immediately removed by the LDAP server. This makes harder to access the entry in your code because you must always check if an attribute key is present before accessing its value. ldap3 helps you to write simpler code because it by default returns an empty attribute even

if it is not present in the LDAP. You can change this behaviour setting to False the `return_empty_attributes` parameter in the Connection object.

### Simple Paged search

The Search operation can perform a *simple paged search* as specified in RFC 2696. The RFC states that you can ask the server to return a specific number of entries in each response set. With every search the server sends back a cookie that you have to provide in each subsequent search. All this information must be passed in a Control attached to the request and the server responds with similar information in a Control attached to the response. ldap3 hides all this machinery in the `paged_search()` function of the **extend.standard** namespace:

```
>>> entries = conn.extend.standard.paged_search('dc=demo1,dc=freeipa,dc=org',
↪'(objectClass=person)', attributes=['cn', 'givenName'], paged_size=5)
>>> for entry in entries:
>>>     print(entry)
```

Entries are returned in a generator, that is better when you have very long list of entries or have memory limitation. Also it sends the requests to the LDAP server only when entries are consumed in the generator. Remember that a generator can be used only one time, so you must elaborate the results in a sequential way. If you don't want the entries returned in a generator you can pass the `generator=False` parameter to get all the entries in a list. In this case all the paged searches are performed by the `paged_search()` function and the set of entries found are queued in a list that is returned.

If you want to directly use the Search operation to perform a Paged search your code should be similar to the following:

```
>>> cookie = "new_cookie"
>>> while cookie:
>>>     conn.search('dc=demo1,dc=freeipa,dc=org', '(objectClass=Person)', attributes=[
↪'cn', 'givenName'], paged_size=5, paged_cookie=cookie)
>>>     cookie = conn.result['controls']['1.2.840.113556.1.4.319']['value']['cookie']
>>>     for entry in conn.entries:
>>>         print(entry)
```

Even in this case the ldap3 library hides the Simple Paged Control machinery but you have to manage the cookie by yourself. The code would be much longer if you would manage directly manage the Simple Search Control. Also you loose the generator feature.

---

**Note:** After performing a traditional LDAP Search operation with a SYNC strategy you get back a collection of Entries in the `entries` property of the Connection object. This collection behaves as the Entries collection of a Reader cursor. For more comprehensive information about the Search operation, see the *SEARCH* documentation. An Entry in the `entries` collection can be modified converting it to a Writable one and applying modifications to it as described in the next chapter.

---

## Tutorial: Database operations

**Warning:** **A more pythonic LDAP**: LDAP operations look clumsy and hard-to-use because they reflect the old-age idea that time-consuming operations should be done on the client to not clutter and hog the server with unneeded elaboration. ldap3 includes a fully functional **Abstraction Layer** that lets you interact with the DIT in a modern and *pythonic* way. With the Abstraction Layer you don't need to directly issue any LDAP operation at all.

In the previous chapter of this tutorial we have tried to access some data in the LDAP database. As any system that stores data, LDAP lets you perform the standard CRUD (Create, Read, Update, Delete) operations, but their usage is someway rudimentary. Again, if you think of the intended use of the original DAP protocol (storing key-values pairs related to an entry in a phone directory) this makes sense: an entry is written once, seldom modified, and eventually deleted, so the create (**Add** in LDAP), update (**Modify** or **ModifyDn**) and delete (**Delete**) operations have a very basic usage while the Read (**Search**) operation is richer in options, but lacks many capabilities you would expect in a modern query language (as 1 to N relationship, joining views, or server data manipulation). Nonetheless almost everything you can do in a modern database can be equally done with LDAP. Furthermore consider that even if an LDAP server can be accessed by multiple clients simultaneously, the LDAP protocol itself has no notion of "transaction", so if you want to issue multiple Add or Modify operations in an atomic way (to keep data consistent), you must investigate the extended operations of the specific LDAP server you're connecting to check if it provides transactions for multiple operations via Controls or Extended operations.

---

**Note:** Synchronous vs Asynchronous: you can submit operations to the server in two different ways: **synchronous** mode and **asynchronous** mode. While with the former you send the request and immediately get the response, in the latter the ldap3 library constantly listens to the server (it uses one independent thread for each connection). When you send a request you must store its *message id* (a unique number that ldap3 stamps on every message of your LDAP session) in your code so you can later query the Connection object for the relevant response when it's ready. You'll probably stick with the synchronous mode, because nowadays LDAP servers are fast to respond, but the asynchronous mode is still useful if your program is event-driven (maybe using an asynchronous event loop).

ldap3 supports both of this models with its different *communication strategies*.

---

LDAP also provides the **Compare** operation that returns True only if an attribute has the value you specify in the request. Even if this operation seems redundant (you could read the attribute and perform the comparison using more powerful tools in your code) you need it to check for the presence of a value (even in a multi-valued attribute) without having the permission to read it. This obviuosly relies upon some "access restriction" mechanism that must be present on the server. LDAP doesn't specify how this mechanism works, so each LDAP server has its specific way of handling authorization. The Compare operation is also used to check the validity of a password (that you can't read) without performing a Bind operation with the specific user.

After any synchronous operation, you'll find the following attributes populated in the Connection object:

- `result`: the result of the last operation (as returned by the server)
- `response`: the entries found (if the last operation is a Search)
- `entries`: the entries found exposed via the ldap3 Abstraction Layer (if the last operation is a Search)
- `last_error`: the error, if any, occurred in the last operation
- `bound`: True if the connection is bound to the server
- `listening`: True if the socket is listening to the server
- `closed`: True if the socket is not open

### Create an Entry

Let's try to add some data to the LDAP DIT:

```
>>> # Create a container for new entries
>>> conn.add('ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', 'organizationalUnit')
True
>>> # Add a new user
```

```
>>> conn.add('cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', 'inetOrgPerson
↪', {'givenName': 'Beatrix', 'sn': 'Young', 'departmentNumber': 'DEV',
↪'telephoneNumber': 1111})
True
```

As you can see we have created a container object and stored a new user in it. You passed the full DN as the first parameter, the objectClass (or objectClasses) as second parameter and a dictonary of attributes as the third parameter. Some attributes are mandatory when adding a new object. You can check the schema to know which are the mandatory attributes you need to provide to successfully create a new object.

Looking at the schema for the *inetOrgPerson* object class we find that there are no mandatory attributes:

```
>>> server.schema.object_classes['inetOrgPerson']
Object class: 2.16.840.1.113730.3.2.2
  Short name: inetOrgPerson
  Superior: organizationalPerson
  May contain attributes: audio, businessCategory, carLicense, departmentNumber,␣
↪displayName, employeeNumber, employeeType, givenName, homePhone, homePostalAddress,␣
↪initials, jpegPhoto, labeledURI, mail, manager, mobile, o, pager, photo, roomNumber,
↪ secretary, uid, userCertificate, x500UniqueIdentifier, preferredLanguage,␣
↪userSMIMECertificate, userPKCS12
  Extensions:
    X-ORIGIN: RFC 2798
```

The *inetOrgPerson* object class is a subclass of the *organizationalPerson* object that again doesn't include any mandatory attributes:

```
>>> server.schema.object_classes['organizationalPerson']
Object class: 2.5.6.7
  Short name: organizationalPerson
  Superior: person
  May contain attributes: title, x121Address, registeredAddress, destinationIndicator,
↪ preferredDeliveryMethod, telexNumber, teletexTerminalIdentifier,␣
↪internationalISDNNumber, facsimileTelephoneNumber, street, postOfficeBox,␣
↪postalCode, postalAddress, physicalDeliveryOfficeName, ou, st, l
  Extensions:
    X-ORIGIN: RFC 4519
  OidInfo: ('2.5.6.7', 'OBJECT_CLASS', 'organizationalPerson', 'RFC4519')
```

The *organizationalPerson* object class is a subclass of the *person* object where we finally find two mandatory attributes:

```
>>> server.schema.object_classes['person']
Object class: 2.5.6.6
  Short name: person
  Superior: top
  Must contain attributes: sn, cn
  May contain attributes: userPassword, telephoneNumber, seeAlso, description
  Extensions:
    X-ORIGIN: RFC 4519
  OidInfo: ('2.5.6.6', 'OBJECT_CLASS', 'person', 'RFC4519')
```

The *person* object class is a subclass of the *top* object. Let's walk up the hierarchy chain:

```
Object class: 2.5.6.0
  Short name: top
  Must contain attributes: objectClass
  Extensions:
```

```
    X-ORIGIN: RFC 4512
  OidInfo: ('2.5.6.0', 'OBJECT_CLASS', 'top', 'RFC4512')
```

*top* is the root of all LDAP classes and defines a single mandatory attributes *objectClass*. Now we know that to successfully create an *inetOrgPerson* we need to provide the *sn*, the *cn* and the *objectClass* attributes at creation time. Let's read the objectClass attribute of the user we created:

```
>>> conn.search('ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', '(cn=*)', attributes=[
↪'objectClass'])
True
>>> conn.entries[0]
DN: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read - READ␣
↪TIME: 2016-10-09T17:36:44.100248
objectClass: inetOrgPerson
             organizationalPerson
             person
             top
```

You can see that *objectClass* is composed of all the hierarchical structure from *inetOrgPerson* to *top*. This means that you can add any of the optional attribute defined in each class of the hierarchy. If you had some *auxiliary* class to the entry you must be sure to satisfy its mandatory attributes.

### Rename an entry

Renaming an entry in LDAP means changing its RDN (*Relative Distinguished Name*) without changing the container where the entry is stored. It is performed with the ModifyDN operation:

```
>>> conn.modify_dn('cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', 'cn=b.
↪smith')
True
```

You have changed the RDN (that in this case uses the *cn* as naming attribute) of the entry from "b.young" to "b.smith". Let's check if the new value is properly stored in the DIT:

```
>>> conn.search('ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', '(cn=b.smith)',␣
↪attributes=['objectclass', 'sn', 'cn', 'givenname'])
True
>>> conn.entries[0]
DN: cn=b.smith,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read - READ␣
↪TIME: 2016-10-11T23:51:28.731000
cn: b.smith
givenname: Beatrix
objectclass: inetOrgPerson
             organizationalPerson
             person
             top
sn: Young
```

As you can see the new *cn* value has been stored in the *cn* attribute. To be consistent in our example we should change the *sn* (surname) from Young to Smith. To achieve this we must wait until we introduce the Modify LDAP operation, the most difficult to use of all the LDAP operations, to update this entry.

### Move entries

ModifyDn is really a two-face operation. You can use it to rename an entry (as in the previous example) or to move an entry to another container. But you cannot perform this two operations together:

```
>>> # Create a container for moved entries
>>> conn.add('ou=moved, ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org',
↪'organizationalUnit')
True
>>> conn.modify_dn('cn=b.smith,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', 'cn=b.
↪smith', new_superior='ou=moved, ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org')
True
```

Quite surprisingly you must provide the very same RDN even if this cannot be changed while moving the object. This could be a problem when moving entries programmatically because you have to break up the DN to its RDNs (remember that each "step" in the DN is really an independent entry with its own RDN.

ldap3 provides the `safe_rdn()` helper function to return the RDN of a DN:

```
>>> from ldap3.utils.dn import safe_rdn
>>> safe_rdn('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org')
[cn=b.smith]
```

Keep in mind that LDAP support a (quite obscure) "multi-rdn" naming option where each part of the RDN is separated with the + character:

```
>>> safe_rdn('cn=b.smith+sn=young,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,
↪dc=org')
['cn=b.smith', 'sn=young']
```

### Update an entry

To change the attributes of an object you must use the Modify operation. There are three kinds of modifications in LDAP: add, delete and replace. **Add** is used to add values to an attribute, and creates the attribute if it doesn't exist. **Delete** deletes values from an attribute and if no values are listed, or if all current values are listed, remove the entire attribute. **Replace** replaces all existing values of an attribute with some new values, creating the attribute if it don't already exist. A replace with no value will delete the entire attribute if it exists, and it is ignored if the attribute doesn't exist.

The hard part in the Modify operation is that you can mix in a single operation the three kinds of modification for a single entry with one or more attributes each with one or more values! So the Modify operation syntax is quite complex: you must provide a DN, a dictionary of attributes and for each attribute a list of modifications where each modification is a tuple with the modification type and the list of values. Let's add a new value to the sn attribute:

```
>>> from ldap3 import MODIFY_ADD, MODIFY_REPLACE, MODIFY_DELETE
>>> conn.modify('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', {
↪'sn': [(MODIFY_ADD, ['Smyth'])]})
True
>>> conn.search('ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', '(cn=b.smith)
↪', attributes=['cn', 'sn'])
True
>>> conn.entries[0]
DN: cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read -␣
↪READ TIME: 2016-10-15T08:35:39.691000
    cn: b.smith
    sn: Young
        Smyth
```

Now remove the old value:

```
>>> conn.modify('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', {
↪'sn': [(MODIFY_DELETE, ['Young'])]})
True
>>> conn.search('ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', '(cn=b.smith)
↪', attributes=['cn', 'sn'])
True
>>> conn.entries[0]
DN: cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read -␣
↪READ TIME: 2016-10-15T08:35:40.331000
    cn: b.smith
    sn: Smyth
```

There is a typo in the previous modify operation (Smyth instead of Smith), let's fix it, replacing values with the right one:

```
>>> conn.modify('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', {
↪'sn': [(MODIFY_REPLACE, ['Smith'])]})
True
>>> conn.search('ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', '(cn=b.smith)
↪', attributes=['cn', 'sn'])
True
>>> conn.entries[0]
DN: cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read -␣
↪READ TIME: 2016-10-15T08:35:40.972000
    cn: b.smith
    sn: Smith
```

Changes in a modify operation can be combined and the syntax of the operation soon becomes complex:

```
>>> conn.modify('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', {
↪'sn': [(MODIFY_ADD, ['Young', 'Johnson']), (MODIFY_DELETE, ['Smith'])], 'givenname
↪': [(MODIFY_REPLACE, ['Mary', 'Jane'])]})
True
>>> conn.search('ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', '(cn=b.smith)
↪', attributes=['cn', 'sn', 'givenName'])
True
>>> conn.entries[0]
DN: cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read -␣
↪READ TIME: 2016-10-15T08:55:47.585000
    cn: b.smith
    givenName: Mary
               Jane
    sn: Young
        Johnson
```

Here you've added 2 values to the *sn* then removed the 'Smith' value from it and replaced the *givenName* with other 2 values, removing all older values.

**Warning:** The MODIFY_REPLACE modification has a misleading name. One could expect it replaces a value with another, but new values only are provided in the Modify operation. What the MODIFY_REPLACE really does is to remove **all** values and add the new values provided. There is no replace at all.

**Note:** The ldap3 Abstraction Layer allows you to use a much more simple and pythonic syntax to achieve the same results.

### Checking attribute values

Very specific to LDAP, and usually not found in other kind of databases, is the **Compare** operation. With this operation you can check if an attribute has a certain value even if you're not able to read it. LDAP doesn't provide a standard authorization access mechanism, so the use of this operation is related to how the vendor has implemented the authorizazion mechanism in the LDAP server you're connecting to.

Let's assume that you don't have the right to read the *departmentNumber* attribute, and you would like to check if the 'b.smith' user is in the 'DEV' department:

```
>>> conn.compare('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org',
↪'departmentNumber', 'DEV')
True
>>> conn.compare('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org',
↪'departmentNumber', 'QA')
False
```

The Compare operation is quite primitive: you can only provide a single attribute and a single value to test against. The operation returns `True` only if one of the values of the attribute is equal to the value provided. Only a single value can be used and no wildcard is allowed.

The only practical use of the Compare operation is when you, as an user with administrative role, want to check the password of another user without actually bind with that user's credentials. In this case you can test the value againts the `userPassword` attribute. Keep in mind the that this only works with the Simple Password authentication method, because for other methods passwords may be stored in a different attribute, or externally to the DIT. Also passwords can (and should) be stored with some encryption mechanism. You must read the documentation of your LDAP server to see if passwords can be successfully checked with the Compare operation.

### What's next

In the next chapter of this tutorial we will start using the **Abstraction Layer**, that hides all the LDAP machinery and let you use standard Python objects to perform the CRUD (Create, Read, Update, Delete) operation that you expect to find in a decent database interface. It uses an **ORM** (*Object Relational Mapper*) to link entries in the DIT with standard Python objects and let you operate on this object in a pythonic way.

Let's move back the 'b.smith* entry to its original context and values and let's create a few more entries in that context:

```
>>> conn.modify_dn('cn=b.smith,ou=moved,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org',
↪ 'cn=b.smith', new_superior='ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org')
True
>>> conn.modify('cn=b.smith,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', {'sn':␣
↪[(MODIFY_DELETE, ['Johnson'])], 'givenname': [(MODIFY_REPLACE, ['Beatrix'])]})
True
>>> conn.modify_dn('cn=b.smith,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', 'cn=b.
↪young')
>>> conn.add('cn=m.johnson,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org',
↪'inetOrgPerson', {'givenName': 'Mary Ann', 'sn': 'Johnson', 'departmentNumber': 'DEV
↪', 'telephoneNumber': 2222})
True
```

```
>>> conn.add('cn=q.gray,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org', 'inetOrgPerson
↪', {'givenName': 'Quentin', 'sn': 'Gray', 'departmentNumber': 'QA', 'telephoneNumber
↪': 3333})
True
```

There should be now three entries in the 'ldap3-tutorial' context. We will use them in the next parts of this tutorial.

## Tutorial: ldap3 Abstraction Layer - Introduction

### A more pythonic LDAP

LDAP was developed in the late '70s when hosts were very expensive. Elaboration was slow and the protocol was developed with the intent of shifting the burden of computing on the client side. So LDAP operations are crude and rough and clients must perform a lot of pre-elaboration before sending their request to servers. This is quite different from what you would expect from any modern API where you send requests to the server (maybe using simple JSON) withouth knowing almost anything about how actually the work is internally done in the server.

---

**Note:** An example of this approach is the Search operation: one would expect that the filter string is simply sent to the server but, if you look at the ldap3 code for the Search operation you'll find a complete parser for the filter, that breaks the filter down to its elemental assertions and builds a recursive representation (similar to an AST, an *Abstract Syntax Tree*) of the filter. It's this representation that is sent to the server in a quite complex binary format (called ASN.1 *Abstract Syntax Notation.1*), not the text of the filter.

---

The ldap3 library includes an Abstraction Layer that lets you interact with the entries in the DIT in a *pythonic way*, with a simple syntax and a consistent behaviour. The Abstraction Layer includes an ORM (Object Relational Mapping) that links Entries (a standard Python class) to entries stored in the DIT. Each Entry object refers to an **ObjectDef** (object class definition) made up of one or more **AttrDef** (attribute type definition) that describes relations between the Attributes stored in the Entry and the attribute values stored for that entry on the LDAP server. With the ORM you can perform all the usual CRUD (Create, Read, Update, Delete) operations, move an entry or rename it. No coding of LDAP operation is actually required.

---

**Note:** In this tutorial we refer to Python objects with an uppercase leading character (Entry, Entries, Attribute, Attributes) words, while refer to objects on the LDAP server with lowercase words (entry, entries, attribute, attributes). Attributes of a generic Python object are referred to as 'property'.

---

With the Abstraction Layer you describe the structure of an LDAP entry and access the LDAP server via a standard Python object, the **Cursor**, that reads and writes Entries from and to the DIT. Optionally you can use a Simplified Query Language in place of the standard LDAP filter syntax.

There are two kinds of Cursor, **Reader** and **Writer**. This mitigates the risk of accidentally changing data in applications that access LDAP only for reading, isolating the writing component: A Reader cursor can't write data to the DIT and a Writer cursor can't read data from it, Writer cursors are only used for modifying the DIT. So reading and writing of data are kept strictly isolated.

Cursors contain Entries. An **Entry** is the Python representation of an entry stored in the LDAP DIT. There are two types of Entries, **Read** and **Writable**. Each Entry has a status that identifies it's current state.

Entries are returned as the result of a LDAP Search operation or of a Reader search operation. Entries are made of Attributes. An **Attribute** is stored in an internal dictionary with case insensitive access and a friendly key. You can access Entry Attributes either as a dictionary or as properties: `entry['CommonName']` is the same of `entry['CommonName]`,''entry.CommonName'', `entry.commonName` and of `entry.commonname` (this feature is helpful when you work at the Python command line. The Abstracton Layer also provides auto-completion of attribute names with

the TAB key). Only Attributes of a Writable Entry can be modified (they actually become WritableAttribute, with updating capability).

Modifications to a Writable Entry are kept in memory until the Entry changes are committed to the DIT. Changes can be discarded before committed. Modifications are declared with the standard *augmented assignments* += and −= or with explicit methods of the WritableAttribute object as `add()`, `set()`, `delete()` and `remove()`.

When creating Entries or assigning new Attribute values, new objects are flagged as **Virtual** until committed, to indicate that they are still not present in the DIT.

Update operations can be applied to a single Entry or to the whole Entry collection of a Writer cursor.

Let's try the same operations we did in the previous chapters of this tutorial. Open the connection to the LDAP server as usual:

```
>>> from ldap3 import Server, Connection, ObjectDef, AttrDef, Reader, Writer, ALL
>>> server = Server('ipa.demo1.freeipa.org', get_info=ALL)
>>> conn = Connection(server, 'uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,
↪dc=org', 'Secret123', auto_bind=True)
```

### Cursor and ObjectDef

ldap3 must know the kind of entry (the LDAP object class) you want to work with to properly manage its attributes. You can take advantage of the schema information read by the Server object and ask the ldap3 library to automatically build an ObjectDef. We can try for the *person* object class, that represents a user in LDAP. The Abstraction Layer will walk up the schema up to the root class reading all the mandatory and optional attributes in its hierarchy, building the AttrDefs collection:

```
>>> obj_person = ObjectDef('person', conn)
```

The `obj_person` object now contains the definition of the LDAP *person* object class as an ObjectDef and includes its attributes as a collection of AttrDef:

```
>>> obj_person
OBJ : person [person (Structural) 2.5.6.6, top (Abstract) 2.5.6.0]
MUST: cn, objectClass, sn
MAY : description, seeAlso, telephoneNumber, userPassword

>>> obj_person.sn
ATTR: sn - mandatory: True - single_value: False
  Attribute type: 2.5.4.4
    Short name: sn, surName
    Single value: False
    Superior: name
    Equality rule: caseIgnoreMatch
    Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
    Mandatory in: person
    Optional in: RFC822localPart, mozillaAbPersonAlpha
    Extensions:
      X-ORIGIN: RFC 4519
      X-DEPRECATED: surName
    OidInfo: ('2.5.4.4', 'ATTRIBUTE_TYPE', ['sn', 'surname'], 'RFC4519')
```

As you can see *person* is a structural class and it's a subclass of the abstract *top* class in the LDAP schema hierarchy. For convenience, mandatory (MUST) Attributes are listed separately from optional (MAY) Attributes because they

are the attributes that must always be present in the entry. You can also access the Attribute definitions as if they were standard properties of the `obj_person` object.

### Entry status

An Entry acquires a number of different statuses in its lifetime and moves from one status to another only when specific events occour. The status of an Entry reflects it's internal state:

Entries created with a Reader cursor can have only one status:

- **Read**: the entry has been read from the DIT and converted to an Entry in the Entries collection.

A Writable Entry in a Writer cursor acquires the following statuses in its lifetime :

- **Writable**: the Entry has been created from a Read one, but no Attribute has been changed yet.
- **Pending changes**: some Attributes have been changed, but still not committed to the LDAP server.
- **Missing mandatory attributes**: Entry misses some mandatory Attribute values, it can't be committed.

There are three global events (delete, move, rename) that locks a Writable Entry until committed (or discarded). In this case the status can be one of the following:

- **Ready for deletion**: Entry is flagged for deletion.
- **Ready for moving**: Entry is flagged for moving.
- **Ready for renaming**: Entry is flagged for renaming.

A new Entry, created in a Writer cursor can have the following status:

- **Virtual**: the Entry is new and still not present in the DIT

After a commit a Writable Entry can be in one of this two statuses:

- **Committed**: changes have been written to the DIT.
- **Deleted**: Entry has been deleted in the DIT.

Note that in a Writable Entry pending changes can be discarded at any time. In this case the Entry status is set to Writable and the original Attribute values are retained.

To get the status of an Entry use the `get_status()` method. You cannot directly change the status of an Entry, it's updated according to the operations performed.

When an Entry is in Pending changes status, new Attributes are flagged as Virtual until committed (or discarded).

## Tutorial: ldap3 Abstraction Layer - Reading data

### Reading entries

Let's define a Reader cursor to get all the entries of class 'inetOrgPerson' in the 'ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org' context:

```
>>> obj_inetorgperson = ObjectDef('inetOrgPerson', conn)
>>> r = Reader(conn, obj_inetorgperson, 'ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org
↪')
>>> r
CURSOR : Reader
CONN   : ldap://ipa.demo1.freeipa.org:389 - cleartext - user: uid=admin,cn=users,
↪cn=accounts,dc=demo1,dc=freeipa,dc=org - not lazy - bound - open - <local: 10.3.9.
↪227:17296 - remote: 209.132.178.99:389> - tls not started - listening -␣
↪SyncStrategy - internal decoder
```

```
DEFS   : ['inetOrgPerson'] [audio, businessCategory, carLicense, cn, departmentNumber,
 ↪ description, destinationIndicator, displayName, employeeNumber, employeeType,␣
 ↪facsimileTelephoneNumber, givenName, homePhone, homePostalAddress, initials,␣
 ↪internationalISDNNumber, jpegPhoto, l, labeledURI, mail, manager, mobile, o,␣
 ↪objectClass, ou, pager, photo, physicalDeliveryOfficeName, postOfficeBox,␣
 ↪postalAddress, postalCode, preferredDeliveryMethod, preferredLanguage,␣
 ↪registeredAddress, roomNumber, secretary, seeAlso, sn, st, street, telephoneNumber,␣
 ↪teletexTerminalIdentifier, telexNumber, title, uid, userCertificate, userPKCS12,␣
 ↪userPassword, userSMIMECertificate, x121Address, x500UniqueIdentifier]
ATTRS  : ['audio', 'businessCategory', 'carLicense', 'cn', 'departmentNumber',
 ↪'description', 'destinationIndicator', 'displayName', 'employeeNumber',
 ↪'employeeType', 'facsimileTelephoneNumber', 'givenName', 'homePhone',
 ↪'homePostalAddress', 'initials', 'internationalISDNNumber', 'jpegPhoto', 'l',
 ↪'labeledURI', 'mail', 'manager', 'mobile', 'o', 'objectClass', 'ou', 'pager', 'photo
 ↪', 'physicalDeliveryOfficeName', 'postOfficeBox', 'postalAddress', 'postalCode',
 ↪'preferredDeliveryMethod', 'preferredLanguage', 'registeredAddress', 'roomNumber',
 ↪'secretary', 'seeAlso', 'sn', 'st', 'street', 'telephoneNumber',
 ↪'teletexTerminalIdentifier', 'telexNumber', 'title', 'uid', 'userCertificate',
 ↪'userPKCS12', 'userPassword', 'userSMIMECertificate', 'x121Address',
 ↪'x500UniqueIdentifier']
BASE   : 'ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org' [SUB]
FILTER : '(objectClass=inetOrgPerson)'
```

We didn't provide any filter, but the Reader automatically uses the ObjectDef class to read entries of the requested object class. Now you can ask the Reader to execute the search, fetching the results in its `entries` property:

```
>>> r.search()
>>> r
CURSOR : Reader
CONN   : ldap://ipa.demo1.freeipa.org:389 - cleartext - user: uid=admin,cn=users,
 ↪cn=accounts,dc=demo1,dc=freeipa,dc=org - not lazy - bound - open - <local: 10.3.9.
 ↪227:27370 - remote: 209.132.178.99:389> - tls not started - listening -␣
 ↪SyncStrategy - internal decoder
DEFS   : ['inetOrgPerson'] [audio, businessCategory, carLicense, cn, departmentNumber,
 ↪ description, destinationIndicator, displayName, employeeNumber, employeeType,␣
 ↪facsimileTelephoneNumber, givenName, homePhone, homePostalAddress, initials,␣
 ↪internationalISDNNumber, jpegPhoto, l, labeledURI, mail, manager, mobile, o,␣
 ↪objectClass, ou, pager, photo, physicalDeliveryOfficeName, postOfficeBox,␣
 ↪postalAddress, postalCode, preferredDeliveryMethod, preferredLanguage,␣
 ↪registeredAddress, roomNumber, secretary, seeAlso, sn, st, street, telephoneNumber,␣
 ↪teletexTerminalIdentifier, telexNumber, title, uid, userCertificate, userPKCS12,␣
 ↪userPassword, userSMIMECertificate, x121Address, x500UniqueIdentifier]
ATTRS  : ['audio', 'businessCategory', 'carLicense', 'cn', 'departmentNumber',
 ↪'description', 'destinationIndicator', 'displayName', 'employeeNumber',
 ↪'employeeType', 'facsimileTelephoneNumber', 'givenName', 'homePhone',
 ↪'homePostalAddress', 'initials', 'internationalISDNNumber', 'jpegPhoto', 'l',
 ↪'labeledURI', 'mail', 'manager', 'mobile', 'o', 'objectClass', 'ou', 'pager', 'photo
 ↪', 'physicalDeliveryOfficeName', 'postOfficeBox', 'postalAddress', 'postalCode',
 ↪'preferredDeliveryMethod', 'preferredLanguage', 'registeredAddress', 'roomNumber',
 ↪'secretary', 'seeAlso', 'sn', 'st', 'street', 'telephoneNumber',
 ↪'teletexTerminalIdentifier', 'telexNumber', 'title', 'uid', 'userCertificate',
 ↪'userPKCS12', 'userPassword', 'userSMIMECertificate', 'x121Address',
 ↪'x500UniqueIdentifier']
BASE   : 'ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org' [SUB]
FILTER : '(objectClass=inetOrgPerson)'
ENTRIES: 3 [executed at: 2016-11-09T09:33:00.342762]
```

There are now three Entries in the Reader. An Entry has some interesting features accessible from its properties and

methods. Because Attribute names are used as Entry properties all the "operational" properties and methods of an Entry start with the **entry_** prefix (the underscore is an invalid character in an attribute name, so there can't be an attribute with that name). It's easy to get a useful representation of an Entry:

```
>>> r[0]
DN: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read - READ␣
↪TIME: 2016-11-09T09:35:02.739203
    cn: b.young
    departmentNumber: DEV
    givenName: Beatrix
    objectClass: inetOrgPerson
                 organizationalPerson
                 person
                 top
    sn: Young
    telephoneNumber: 1111
```

Let's explore some of them:

```
>>> # get the DN of an entry
>>> r[0].entry_dn
'cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org'

>>> # query the attributes in the Entry as a list of names
>>> r[0].entry_attributes
['destinationIndicator', 'x500UniqueIdentifier', 'audio', 'photo', 'uid', 'l', 'pager
↪', 'carLicense', 'street', 'teletexTerminalIdentifier', 'o', 'st',
↪'homePostalAddress', 'preferredDeliveryMethod', 'roomNumber', 'sn', 'homePhone',
↪'x121Address', 'displayName', 'userSMIMECertificate', 'userPassword', 'title',
↪'physicalDeliveryOfficeName', 'mail', 'initials', 'ou', 'businessCategory', 'seeAlso
↪', 'jpegPhoto', 'registeredAddress', 'facsimileTelephoneNumber', 'postalAddress',
↪'telephoneNumber', 'mobile', 'labeledURI', 'postalCode', 'objectClass',
↪'employeeNumber', 'secretary', 'employeeType', 'description', 'cn', 'userCertificate
↪', 'userPKCS12', 'postOfficeBox', 'departmentNumber', 'givenName',
↪'internationalISDNNumber', 'preferredLanguage', 'telexNumber', 'manager']

>>> # query the attributes in the Entry as a dict of key/value pairs
>>> r[0].entry_attributes_as_dict
{'destinationIndicator': [], 'x500UniqueIdentifier': [], 'audio': [], 'photo': [],
↪'uid': [], 'l': [], 'pager': [], 'carLicense': [], 'street': [],
↪'teletexTerminalIdentifier': [], 'o': [], 'homePostalAddress': [],
↪'preferredDeliveryMethod': [], 'roomNumber': [], 'st': [], 'homePhone': [],
↪'x121Address': [], 'displayName': [], 'userSMIMECertificate': [], 'userPassword':␣
↪[], 'title': [], 'physicalDeliveryOfficeName': [], 'mail': [], 'preferredLanguage':␣
↪[], 'initials': [], 'internationalISDNNumber': [], 'ou': [], 'businessCategory': [],
↪ 'seeAlso': [], 'jpegPhoto': [], 'registeredAddress': [], 'facsimileTelephoneNumber
↪': [], 'postalAddress': [], 'telephoneNumber': ['1111'], 'mobile': [], 'labeledURI
↪': [], 'postalCode': [], 'objectClass': ['inetOrgPerson', 'organizationalPerson',
↪'person', 'top'], 'employeeNumber': [], 'description': [], 'employeeType': [],
↪'secretary': [], 'cn': ['b.young'], 'userPKCS12': [], 'postOfficeBox': [],
↪'departmentNumber': ['DEV'], 'givenName': ['Beatrix'], 'sn': ['Young'],
↪'userCertificate': [], 'telexNumber': [], 'manager': []}


>>> # let's check which attributes are mandatory
>>> r[0].entry_mandatory_attributes
['sn', 'objectClass', 'cn']
```

```
>>> # convert the Entry to LDIF
>>> print(r[0].entry_to_ldif())
version: 1
dn: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: person
objectClass: top
sn: Young
telephoneNumber: 1111
cn: b.young
departmentNumber: DEV
givenName: Beatrix
# total number of entries: 1

>>> print(r[0].entry_to_json(include_empty=False))  # Use include_empty=True to␣
↪include empty attributes
{
    "attributes": {
        "cn": [
            "b.young"
        ],
        "departmentNumber": [
            "DEV"
        ],
        "givenName": [
            "Beatrix"
        ],
        "objectClass": [
            "inetOrgPerson",
            "organizationalPerson",
            "person",
            "top"
        ],
        "sn": [
            "Young"
        ],
        "telephoneNumber": [
            "1111"
        ]
    },
    "dn": "cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org"
}
```

If you search for the uid=admin entry there are some auxiliary classes attached to it. The uid=admin entry is not an *inetOrgPerson* but a *person*, so you must use the `obj_person` defined in the previous chapter of this tutorial:

```
>>> obj_person
OBJ : person [person (Structural) 2.5.6.6, top (Abstract) 2.5.6.0]
MUST: cn, objectClass, sn
MAY : description, seeAlso, telephoneNumber, userPassword
```

This ObjectDef lacks the *uid* attributes, used for naming the admin entry, so we must add it to the Object definition:

```
>>> obj_person += 'uid'  # implicitly creates a new AttrDef
>>> obj_person
OBJ : person [person (Structural) 2.5.6.6, top (Abstract) 2.5.6.0]
MUST: cn, objectClass, sn
```

```
MAY : description, seeAlso, telephoneNumber, uid, userPassword
```

Now let's build the Reader cursor, using the Simplified Query Language, note how the filter is converted:

```
>>> r = Reader(conn, obj_person, 'cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org',
 ↪'uid:=admin')
>>> r
CURSOR : Reader
CONN   : ldap://ipa.demo1.freeipa.org:389 - cleartext - user: uid=admin,cn=users,
 ↪cn=accounts,dc=demo1,dc=freeipa,dc=org - not lazy - bound - open - <local: 10.3.9.
 ↪227:27438 - remote: 209.132.178.99:389> - tls not started - listening - ␣
 ↪SyncStrategy - internal decoder
DEFS   : ['person'] [cn, description, objectClass, seeAlso, sn, telephoneNumber, uid, ␣
 ↪userPassword]
ATTRS  : ['cn', 'description', 'objectClass', 'seeAlso', 'sn', 'telephoneNumber', 'uid
 ↪', 'userPassword']
BASE   : 'cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org' [SUB]
QUERY  : 'uid:=admin' [AND]
PARSED : 'uid: =admin' [AND]
FILTER : '(&(objectClass=person)(uid=admin))'
```

**And finally perform the search operation::**

```
>>> r.search()
[DN: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS: Read - ␣
 ↪READ TIME: 2016-11-09T09:59:56.393112
    cn: Administrator
    objectClass: top
                 person
                 posixaccount
                 krbprincipalaux
                 krbticketpolicyaux
                 inetuser
                 ipaobject
                 ipasshuser
                 ipaSshGroupOfPubKeys
                 ipaNTUserAttrs
    sn: Administrator
    uid: admin]
```

Only one entry is found. As you can see this Entry has additional auxiliary object classes attached. This means that there can be other attributes stored in the entry. Let's define an ObjectDef that also requests the 'posixAccount' and the 'krbprincipalaux' object classes:

```
>>> obj_person = ObjectDef(['person', 'posixAccount', 'krbprincipalaux'], conn)
OBJ : person, posixAccount, krbPrincipalAux [person (Structural) 2.5.6.6, top ␣
 ↪(Abstract) 2.5.6.0, posixAccount (Auxiliary) 1.3.6.1.1.1.2.0, top (Abstract) 2.5.6.
 ↪0, krbPrincipalAux (Auxiliary) 2.16.840.1.113719.1.301.6.8.1]
MUST: cn, gidNumber, homeDirectory, objectClass, sn, uid, uidNumber
MAY : description, gecos, krbAllowedToDelegateTo, krbCanonicalName, krbExtraData, ␣
 ↪krbLastAdminUnlock, krbLastFailedAuth, krbLastPwdChange, krbLastSuccessfulAuth, ␣
 ↪krbLoginFailedCount, krbPasswordExpiration, krbPrincipalAliases, ␣
 ↪krbPrincipalAuthInd, krbPrincipalExpiration, krbPrincipalKey, krbPrincipalName, ␣
 ↪krbPrincipalType, krbPwdHistory, krbPwdPolicyReference, krbTicketPolicyReference, ␣
 ↪krbUPEnabled, loginShell, seeAlso, telephoneNumber, userPassword
```

As you can see the ObjectDef now includes all Attributes from the *person*, *top*, *posixAccount* and *krbPrincipalAux* classes. Now create a new Reader, its filter will automatically includes all the requested object classes:

```
>>> r = Reader(conn, obj_person, 'dc=demo1,dc=freeipa,dc=org', 'uid:=admin')
>>> r
CURSOR : Reader
CONN   : ldap://ipa.demo1.freeipa.org:389 - cleartext - user: uid=admin,cn=users,
 ↪cn=accounts,dc=demo1,dc=freeipa,dc=org - not lazy - bound - open - <local: 10.3.9.
 ↪227:29283 - remote: 209.132.178.99:389> - tls not started - listening -␣
 ↪SyncStrategy - internal decoder
DEFS   : ['person', 'posixAccount', 'krbPrincipalAux'] [cn, description, gecos,␣
 ↪gidNumber, homeDirectory, krbAllowedToDelegateTo, krbCanonicalName, krbExtraData,␣
 ↪krbLastAdminUnlock, krbLastFailedAuth, krbLastPwdChange, krbLastSuccessfulAuth,␣
 ↪krbLoginFailedCount, krbPasswordExpiration, krbPrincipalAliases,␣
 ↪krbPrincipalAuthInd, krbPrincipalExpiration, krbPrincipalKey, krbPrincipalName,␣
 ↪krbPrincipalType, krbPwdHistory, krbPwdPolicyReference, krbTicketPolicyReference,␣
 ↪krbUPEnabled, loginShell, objectClass, seeAlso, sn, telephoneNumber, uid, uidNumber,
 ↪ userPassword]
ATTRS  : ['cn', 'description', 'gecos', 'gidNumber', 'homeDirectory',
 ↪'krbAllowedToDelegateTo', 'krbCanonicalName', 'krbExtraData', 'krbLastAdminUnlock',
 ↪'krbLastFailedAuth', 'krbLastPwdChange', 'krbLastSuccessfulAuth',
 ↪'krbLoginFailedCount', 'krbPasswordExpiration', 'krbPrincipalAliases',
 ↪'krbPrincipalAuthInd', 'krbPrincipalExpiration', 'krbPrincipalKey',
 ↪'krbPrincipalName', 'krbPrincipalType', 'krbPwdHistory', 'krbPwdPolicyReference',
 ↪'krbTicketPolicyReference', 'krbUPEnabled', 'loginShell', 'objectClass', 'seeAlso',
 ↪'sn', 'telephoneNumber', 'uid', 'uidNumber', 'userPassword']
BASE   : 'dc=demo1,dc=freeipa,dc=org' [SUB]
QUERY  : 'uid:=admin' [AND]
PARSED : 'uid: =admin' [AND]
FILTER : '(&(&
 ↪(objectClass=person)(objectClass=posixAccount)(objectClass=krbPrincipalAux))(uid=admin))
 ↪'

>>> r.search()
>>> r[0]
DN: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS: Read - READ␣
 ↪TIME: 2016-11-09T10:03:47.741382
    cn: Administrator
    gecos: Administrator
    gidNumber: 1120000000
    homeDirectory: /home/admin
    krbExtraData: b'\x00\x02\xd2\xad"Xroot/admin@DEMO1.FREEIPA.ORG\x00'
    krbLastFailedAuth: 2016-11-09 06:22:15+00:00
    krbLastPwdChange: 2016-11-09 05:02:10+00:00
    krbLastSuccessfulAuth: 2016-11-09 09:03:49+00:00
    krbLoginFailedCount: 0
    krbPasswordExpiration: 2017-11-09 05:02:10+00:00
    krbPrincipalName: admin@DEMO1.FREEIPA.ORG
    loginShell: /bin/bash
    objectClass: top
                 person
                 posixaccount
                 krbprincipalaux
                 krbticketpolicyaux
                 inetuser
                 ipaobject
                 ipasshuser
                 ipaSshGroupOfPubKeys
                 ipaNTUserAttrs
    sn: Administrator
    uid: admin
```

```
    uidNumber: 1120000000
```

Note that Attribute are properly formatted thanks to information read in the server schema. For example the krbLastPwdChange is stored as a date (Generalized Time, a standard LDAP data type):

```
>>> obj_person.krblastpwdchange
ATTR: krbLastPwdChange - mandatory: False - single_value: True
  Attribute type: 2.16.840.1.113719.1.301.4.45.1
    Short name: krbLastPwdChange
    Single value: True
    Equality rule: generalizedTimeMatch
    Syntax: 1.3.6.1.4.1.1466.115.121.1.24 [('1.3.6.1.4.1.1466.115.121.1.24', 'LDAP_
↪SYNTAX', 'Generalized Time', 'RFC4517')]
    Optional in: krbPrincipalAux
```

So the ldap3 library returns it as a DateTime object (with time zone info):

```
>>> type(r[0].krblastpwdchange.value)
<class 'datetime.datetime'>
```

**Warning:** The ldap3 library returns dates with Time Zone info. These dates can be compared only with dates with Time Zone. You can't compare them with a "naive" date object.

**Note:** Attributes have three properties for getting their values: the `values` property returns always a list containing all values (even in a single-valued attribute; the `value` property returns the very same list in a multi-valued attribute or the value in a single-valued attribute. `raw_attributes` always returns a list of the binary values received in the LDAP response. When the schema is available the `values` and `value` properties are properly formatted as standard Python types. You can add additional custom formatters with the `formatter` parameter of the Server object.

If you look at the raw data read from the server, you get the values actually stored in the DIT:

```
>>> r[0].krblastpwdchange.raw_values
[b'20161109050210Z']
```

Similar formatting is applied to other well-known attribute types, for example GUID or SID in Active Directory. Numbers are returned as `int`:

```
>>> e[0].krbloginfailedcount.value
krbLoginFailedCount: 0
>>> type(e[0].krbloginfailedcount.value)
<class 'int'>
>>> e[0].krbloginfailedcount.raw_values
[b'0']
```

### Search scope

By default the Reader searches the whole sub tree starting from the specified base. If you want to search entries only in the base, you can pass the `sub_tree=False` parameter in the Reader definition. You can also override the default scope with the `search_level()`, `search_object()` and `search_subtree()` methods of the Reader object:

```
>>> r.search_level()  # search only at the 'dc=demo1,dc=freeipa,dc=org' context
>>> print(len(r))  # the admin entry in in the cn=users,cn=account container, so no␣
↪entry is found
0
>>> r.search_subtree()  # search walking down from the 'dc=demo1,dc=freeipa,dc=org'␣
↪context
>>> print(len(r))
1
```

### Matching entries in cursor results

Once a cursor is populated with entries you can get a specific entry with the standard index feature of List object: `r.entries[0]` returns the first entry found, `r.entries[1]` returns he second one and any subsequent entry is returned by the relevant index number. The Cursor object has a shortcut for this operation: you can use `r[0]`, `r[1]` (and so on) to perform the same operation. Furthermore, the Cursor object has an useful feature that helps you to find a specific entry without knowing its index: when you use a string as the Cursor index the text will be searched in all entry DNs. If only one entry matches it is returned, if more than one entry match the text a KeyError exception is raised. You can also use the `r.match_dn(dn)` method to return all entries with the specified text in the DN and `r.match(attributes, value)` to return all entries that contain the `value` in any of the specified `attributes` where you can pass a single attribute name or a list of attribute names. When searching for values the either the formatted attribute and the raw value are checked.

## Tutorial: ldap3 Abstraction Layer - Writing data

### Writing entries

Modifying data on an LDAP server is easy with the Abstraction Layer, just add a new Entry with its Attribute values (or change the Attribute values of an existing Entry) and commit the pending changes to the DIT via the Writer cursor. You can obtain a Writer cursor in a number of ways:

- from a Reader Cursor, using the `Writer.from_cursor()` static method that populates the `entries` collection with a copy of the Entries from the Reader cursor.
- from a Search response, using the `Writer.from_response()` static method, that populates the `entries` collection with a copy of the Entries from the Search response.
- from a single Entry in a Reader cursor, using the `entry_writable()` method of the Entry, that returns a new Writable Entry (and also creates its Writer cursor).
- from a single Entry in a Search response, using the `entry_writable()` method of the Entry, that returns a new Writable Entry (and also creates its Writer cursor)
- as a new instance of the ldap3.abstract.Writer class, using `Writer()` that creates a new Writer cursor with an empty `entries` collection. With this cursor you can only create new Entries.

Let's obtain a Writer cursor from the inetOrgPerson Reader we used in the previous chapter:

```
>>> r = Reader(conn, obj_inetorgperson, 'ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org
↪')
>>> r.search()
>>> w = Writer.from_cursor(r)
>>> w
CURSOR : Writer
CONN   : ldap://ipa.demo1.freeipa.org:389 - cleartext - user: uid=admin,cn=users,
↪cn=accounts,dc=demo1,dc=freeipa,dc=org - not lazy - bound - open - <local: 10.3.9.
↪227:29872 - remote: 209.132.178.99:389> - tls not started - listening -␣
↪SyncStrategy - internal decoder
```

```
DEFS   : ['inetOrgPerson'] [audio, businessCategory, carLicense, cn, departmentNumber,
 ↪ description, destinationIndicator, displayName, employeeNumber, employeeType,␣
 ↪facsimileTelephoneNumber, givenName, homePhone, homePostalAddress, initials,␣
 ↪internationalISDNNumber, jpegPhoto, l, labeledURI, mail, manager, mobile, o,␣
 ↪objectClass, ou, pager, photo, physicalDeliveryOfficeName, postOfficeBox,␣
 ↪postalAddress, postalCode, preferredDeliveryMethod, preferredLanguage,␣
 ↪registeredAddress, roomNumber, secretary, seeAlso, sn, st, street, telephoneNumber,␣
 ↪teletexTerminalIdentifier, telexNumber, title, uid, userCertificate, userPKCS12,␣
 ↪userPassword, userSMIMECertificate, x121Address, x500UniqueIdentifier]
ATTRS  : ['audio', 'businessCategory', 'carLicense', 'cn', 'departmentNumber',
 ↪'description', 'destinationIndicator', 'displayName', 'employeeNumber',
 ↪'employeeType', 'facsimileTelephoneNumber', 'givenName', 'homePhone',
 ↪'homePostalAddress', 'initials', 'internationalISDNNumber', 'jpegPhoto', 'l',
 ↪'labeledURI', 'mail', 'manager', 'mobile', 'o', 'objectClass', 'ou', 'pager', 'photo
 ↪', 'physicalDeliveryOfficeName', 'postOfficeBox', 'postalAddress', 'postalCode',
 ↪'preferredDeliveryMethod', 'preferredLanguage', 'registeredAddress', 'roomNumber',
 ↪'secretary', 'seeAlso', 'sn', 'st', 'street', 'telephoneNumber',
 ↪'teletexTerminalIdentifier', 'telexNumber', 'title', 'uid', 'userCertificate',
 ↪'userPKCS12', 'userPassword', 'userSMIMECertificate', 'x121Address',
 ↪'x500UniqueIdentifier']
ENTRIES: 3 [executed at: 2016-11-09T14:24:49.374675]
```

Entries in a Writer cursor are standard Python object, so you can modify them with standard Python code:

```
>>> w[0]
DN: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Writable - READ␣
 ↪TIME: 2016-11-09T14:26:03.866351
    cn: b.young
    departmentNumber: DEV
    givenName: Beatrix
    objectClass: inetOrgPerson
                 organizationalPerson
                 person
                 top
    sn: Young
    telephoneNumber: 1111
```

The entry is in **Writable** status now, so you can try to update some values. All modifications are stored in memory until committed or the entry is returned to its original values:

```
>>> w[0].sn += 'Smyth'  # Add 'Smith' value from the sn
>>> w[0].sn += 'Johnson'  # Add 'Johnson' value from the sn
>>> w[0].sn -= 'Young'  # remove the 'Young' value from the sn
```

Now let's revise the modifications we have requested:

```
>>> w[0].entry_changes
OrderedDict([('sn', [('MODIFY_ADD', ['Smyth']), ('MODIFY_ADD', ['Johnson']), ('MODIFY_
 ↪DELETE', ['Young'])])])
```

Modifications to an Entry are stored in a way (OrderedDict) that preserves the insertion sequence. This can be helpful with specific LDAP operations that request that an attribute is modified before an other one in the same LDAP operation

We made a typo so discard the changes and insert the correct values:

```
>>> w[0].sn.discard()
>>> w[0].sn += ['Smith', 'Johnson']  # add a list of values
>>> w[0].sn -= 'Young'  # remove the 'Young' value from the sn
```

```
>>> w[0]
DN: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Writable,␣
↪Pending changes - READ TIME: 2016-11-09T14:30:43.181520
    cn: b.young
    departmentNumber: DEV
    givenName: Beatrix
    objectClass: inetOrgPerson
                 organizationalPerson
                 person
                 top
    sn: Young
        CHANGES: [('MODIFY_ADD', ['Smith', 'Johnson']), ('MODIFY_DELETE', ['Young'])]
    telephoneNumber: 1111
```

Entry status is set to *Writable, Pending changes*, this means that mandatory Attributes are set and the Entry can be written in the DIT:

```
>>> w.commit()  # commit all entries with pending changes
>>> w[0]
DN: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Writable,␣
↪Committed - READ TIME: 2016-11-09T14:32:14.377498
    cn: b.young
    departmentNumber: DEV
    givenName: Beatrix
    objectClass: inetOrgPerson
                 organizationalPerson
                 person
                 top
    sn: Smith
        Johnson
    telephoneNumber: 1111
```

Entry has been written on the DIT and its state is now *Writable, Committed*. If you look at the original Entry in the Reader you will find that it's been updated with the new values:

```
>>> r[0]
DN: cn=b.young,ou=ldap3-tutorial,dc=demo1,dc=freeipa,dc=org - STATUS: Read - READ␣
↪TIME: 2016-11-09T14:32:14.377498
    cn: b.young
    departmentNumber: DEV
    givenName: Beatrix
    objectClass: inetOrgPerson
                 organizationalPerson
                 person
                 top
    sn: Smith
        Johnson
    telephoneNumber: 1111
```

Refreshing of the original Entry is triggered only if both cursors are using the same Server object. If you use the Writer cursor to copy Entries to another LDAP server refreshing of the original Entry is not executed.

For specific types (boolean, integers and dates) you can set the value to the relevant Python type. The ldap3 library will perform the necessary conversion to the value expected from the LDAP server.

## ldap3 Features

1. ldap3 strictly conforms to the current RFCs for the version 3 of the LDAP protocol (from 4510 to 4519):
    - RFC4510: Technical Specification Road Map
    - RFC4511: The Protocol
    - RFC4512: Directory Information Models
    - RFC4513: Authentication Methods and Security Mechanisms
    - RFC4514: String Representation of Distinguished Names
    - RFC4515: String Representation of Search Filters
    - RFC4516: Uniform Resource Locator
    - RFC4517: Syntaxes and Matching Rules
    - RFC4518: Internationalized String Preparation
    - RFC4519: Schema for User Applications

    The following RFCs, describing additional functionalities of the LDAP3 protocol, are also followed:
    - RFC2696: LDAP Control Extension for Simple Paged Results Manipulation
    - RFC2849: The LDAP Data Interchange Format (LDIF) - Technical Specification
    - RFC3045: Storing Vendor Information in the LDAP root DSE
    - RFC3062: LDAP Password Modify Extended Operation
    - RFC4525: Modify-Increment Extension
    - RFC4530: entryUUID Operational Attribute
    - RFC4532: "Who am I?" Operation
    - RFC5020: entryDN Operational Attribute
2. Platform independent (tested on Linux and Windows) architecture:
    - The library **runs on Windows, Linux, FreeBSD, OpenBSD, and Mac OSX** and (possibly) on other systems where it can gain access to the network via a Python interpreter and its Standard Library.
3. Based on **pure Python code**:
    - No need to install binaries or non Python code. The very same code works on Windows, Linux, Mac OS X, FreeBSD, OpenBSD and other systems, either in Python 2 or Python 3.
    - ldap3 **doesn't need a C compiler neither the OpenLDAP library**.
    - The library is self-contained and its installation is the same on any platform.
    - Socket and thread programming is appropriate for the platform used, with no changes needed in the configuration and in the exposed API.
    - The ldap3 library depends on the standard Python library and the pyasn1 package only. If you need Kerberos support you must install the *gssapi* package. ldap3 includes a backport (from Python 3.4.3) of `` `ssl.check_hostnames` `` to be used on older (version < 2.7.10) Python versions. If you want to use a more up to date version of the check_hostnames feature you can install the *backports.ssl_check_hostnames* package that should be kept updated with the Standard Library of the latest Python release by its maintainers.
4. Compatible with Python 2 and Python 3:

- A **single codebase** for Python 2 and Python 3
- Developed in **Python 3 native code** that works in Python 2 too.
- The library is **compatible with Python 2** (2.6 and 2.7) without the need of any code compatibility parser/converter.
- Testing is done in Python 3 (3.4, 3.5) Python 2 (2.6, 2.7), PyPy and PyPy3
- Unicode strings are properly managed in each Python version.

5. Multiple *connection strategies* to choose from, either synchronous or asynchronous:
   - The library has different ways to connect to the LDAP server (no-thread, single-threaded, multi-threaded). This is achieved with **pluggable communication strategies** that can be changed on a per-connection basis.
   - SYNC, ASYNC, LDIF, RESTARTABLE (fault-tolerant), REUSABLE (fault-tolerant and pooled), are currently defined.
   - MOCK_SYNC strategy to emulate an LDAP server without connecting to a real server. Useful for testing your application
6. Simplified query construction language:
   - The library includes an optional fully functin **Abstraction Layer** for performing LDAP operations. With the Abstraction Layer you don't need to directly issue any LDAP operation at all.
7. Clear or secured access
   - ldap3 allows plaintext (**ldap:**), secure (**ldaps:**) and UNIX socket (**ldapi:**) access to the LDAP server.
   - The NTLM access method is available to connect to Active Directory servers using NTLM v2 authentication.

# Installation and configuration

Installation is straightforward and can be done via a package manager or from the source.

## Installation with a package manager

You need the **pip** package (or another package manager that can download and install from pyPI) to install ldap3. Then you can download and install the ldap3 library directly from pyPI:

```
pip install ldap3
```

This library has only one dependence on the *pyasn1* module, You can install it or let the installer do it for you.

If you need to access a server with the Kerberos SASL authentication mechanism you must install the *gssapi* package.

ldap3 includes a backport (from Python 3.4.3) of `ssl.check_hostnames` to be used on older (version < 2.7.10) Python version. If you want to use a more up to date version of the check_hostnames feature you can install the *backports.ssl_check_hostnames* package that should be kept updated with the Standard Library of the latest Python release by its maintainers.

## Installation from the source

You can download the latest source from https://github.com/cannatag/ldap3 then you can install the library with:

```
python setup.py install
```

## Global configuration

in the **ldap3.utils.config** package there are some configurable settings:

- POOLING_LOOP_TIMEOUT = 10 # number of seconds to wait before restarting a cycle to find an active server in the pool
- RESPONSE_SLEEPTIME = 0.05 # seconds to wait while waiting for a response in asynchronous strategies
- RESPONSE_WAITING_TIMEOUT = 3 # waiting timeout for receiving a response in asynchronous strategies
- SOCKET_SIZE = 4096 # socket byte size
- CHECK_AVAILABILITY_TIMEOUT = 2.5 # default timeout for socket connect when checking availability
- RESET_AVAILABILITY_TIMEOUT = 5 # default timeout for resetting the availability status when checking candidate addresses
- RESTARTABLE_SLEEPTIME = 2 # time to wait in a restartable strategy before retrying the request
- RESTARTABLE_TRIES = 30 # number of times to retry in a restartable strategy before giving up. Set to True for unlimited retries
- REUSABLE_THREADED_POOL_SIZE = 5
- REUSABLE_THREADED_LIFETIME = 3600 # 1 hour
- DEFAULT_THREADED_POOL_NAME = 'REUSABLE_DEFAULT_POOL'
- ADDRESS_INFO_REFRESH_TIME = 300 # seconds to wait before refreshing address info from dns
- ADDITIONAL_ENCODINGS = ['latin-1'] # some broken LDAP implementation may have different encoding than those expected by RFCs
- IGNORE_MALFORMED_SCHEMA = False # some flaky LDAP servers returns malformed schema. If True no expection is raised and schema is thrown away

This parameters are library-wide and usually you should keep the default values.

You can use the `get_config_parameter()` and `set_config_parameter()` functions in the ldap3 namespace to get and set the configurable parameters at runtime.

## Importing objects and constants

All objects and constants needed to use the ldap3 library can be imported from the **ldap3** namespace:

```
from ldap3 import Connection, Server, ANONYMOUS, SIMPLE, SYNC, ASYNC
```

## Library errors

You can deal with errors in two different ways. By default in synchronous strategies each LDAP operation returns a True/False value that specify if the operation has been successful or not. In case of failures you can check the error description in the `` `last_error` `` attribute of the Connection object. In some cases an exception of the custom hierarchy starting from the `` `LDAPExceptionError` `` class is raised with a description of the error condition in the *args* attribute.

If you prefer to deal always with Exceptions you can set the `raise_exceptions` attribute to `True` in the Connection object definition. From now on the Connection will raise exceptions for all operations that return result codes different from `RESULT_SUCCESS`, `RESULT_COMPARE_FALSE`, `RESULT_COMPARE_TRUE`, `RESULT_REFERRAL`.

Communication exceptions have multiple inheritance either from `` `LDAPCommunicationError` `` and the specific socket exception.

Exceptions are defined in the **ldap3.core.exceptions** package.

# Server

## Server object

The Server object specifies the DSA (Directory Server Agent) LDAP server that will be used by the connection. To create a new Server object the following parameters are available:

- host: name or ip or the complete url in the scheme://hostname:hostport format of the server (required) - port and scheme (ldap or ldaps) defined here have precedence over the parameters port and use_ssl
- port: the port where the DSA server is listening (defaults to 389, for a cleartext connection, 636 for a secured connection)
- use_ssl: specifies if the connection is on a secure port (defaults to False). When True the secure port is usually set to 636.
- allowed_referral_hosts: specifies which servers are considered reliable as referrals (defaults to None)
  - Format is a list of tuples; [(server, allow_auth), (server, allow_auth), ...]
  - server is an IP address or DNS name. Specify an asterisk (*) to accept any server.
  - allow_auth is a boolean to indicate if authentication to that server is allowed; if False only anonymous bind will be used.
- get_info: specifies if the server schema and server specific info must be read (defaults to SCHEMA). Possible values are:
  - NONE: no information is gathered from the server
  - DSA: server information is stored in server.info
  - SCHEMA: schema information is stored in server.schema
  - ALL: server and schema information are gathered and stored in server.info and server.schema
  - OFFLINE_EDIR_8_8_8: pre-built schema and info for NetIQ eDirectory 8.8.8
  - OFFLINE_AD_2012_R2: pre-built schema and info for Microsoft Active Directory from Windows Server 2012 R2
  - OFFLINE_SLAPD_2_4: pre-built schema and info for Openldap 2.4
  - OFFLINE_DS389_1_3_3: pre-built schema and info for DS389 1.3.3
- mode: specifies dual IP stack behaviour for resolving LDAP server names in DNS: Possible values are:
  - IP_SYSTEM_DEFAULT: disable dual stack feature. Use system default
  - IP_V4_ONLY: use only IPV4 names
  - IP_V6_ONLY: use only IPV6 names

  - IP_V4_PREFERRED: tries IPV4 names and if connection fails tries IPV6
  - IP_V6_PREFERRED: tries IPV6 names and if connection fails tries IPV4
- tls: Tls object that contains information about the certificates and the trusted roots needed to establish a secure connection (defaults to None). If None any server certificate will be accepted.
- formatter: a dictionary of custom formatter for attributes returned in search
- connect_timeout: timeout in seconds for the connect operation

Example:

```
server = Server('server1', port=636, use_ssl=True, allowed_referral_hosts=[('server2',
↪ True), ('server3', False)])
```

A server can be implicitly defined with default directly in the Connection definition:

```
connection = Connection('server1', user='cn=user1,o=test', password='password')
```

## Server Pool

**Note:** Active strategies

Active strategies check if the server is listening on the specified port. When the 'active' attribute is set to True the strategy tries to open and close a socket on the port. If your LDAP server has problems with the opening and closing of sockets you can set 'active' to False..

Different Server objects can be grouped in a ServerPool object. A ServerPool object can be specified in the Connection object to obtain an high availability (HA) connection. This is useful for long standing connections (for example an LDAP authenticator module in an application server) or when you have a multi replica LDAP server infrastructure. The `active` and `exhaust` parameter accept either a boolean or a number: if you set `active=True` while defining the ServerPool the strategy will check for server availability, you can also set this attribute to the maximum number of cycles to try before giving up with an LDAPServerPoolExhaustedError exception. With `exhaust=True` if a server is not active it will be removed by the pool, if you set it to a number this will be the number of seconds an unreachable server is considered offline. WHen this timout expires the server is reinserted in the pool and checked again for availability.

When all servers in a pool are not available the strategy will wait for the number of seconds specified in `ldap.POOLING_LOOP_TIMEOUT` before starting a new cycle. This defaults to 10 seconds.

The pool can have different HA strategies:

- FIRST: gets the first server in the pool, if 'active' is set to True gets the first available server
- ROUND_ROBIN: each time the connection is open the subsequent server in the pool is used. If active is set to True unavailable servers will be discarded
- RANDOM: each time the connection is open a random server is chosen in the pool. If active is set to True unavailable servers will be discarded

A server pool can be defined in different ways:

```
server1 = Server('server1')
server2 = Server('server2')
server3 = Server('server1', port=636, use_ssl=True)
```

- explicitly with Server objects in the init:

```
server_pool = ServerPool([server1, server2, server3], POOLING_STRATEGY_ROUND_
↪ROBIN, active=True, exhaust=True)
```

- explicitly with an add operation in the pool object:

```
server_pool = ServerPool(None, POOLING_STRATEGY_ROUND_ROBIN_ACTIVE)
server_pool.add(server1)
server_pool.add(server2)
server_pool.add(server3)
```

- implicitly directly in the Connection object init (passing a list of servers):

```
conn = Connection([server1, server2, server3])  # the ServerPool object is␣
↪defined with the default pooling strategy
```

Pools can be dynamically changed. You can add and remove Server objects from pools even if they are already used in Connection:

```
server4 = Server('server2', port=636, use_ssl=True)
server_pool.remove(server2)
server_pool.add(server4)
```

Connections are notified of the change and can reopen the socket to the new server at next open() operation.

Custom formatters can be used to specify how an attribute value must be returned in the 'attributes' attribute of the search entry object. A formatter must be a callable that receives a bytes value and return an object. The object will be returned in the 'attributes' if the schema is read and check_names connection parameter is True. If the attribute is defined in the schema as 'multi_value' the attribute value is returned as a list (even if only a single value is present) else it's returned as a single value.

## Offline Schema

If your LDAP server doesn't return the DSA info or the Schema you can load pre-built schemas and infos with the get_info parameter. Schemas are available for eDirectory, Active Directory and Openldap.

You can also save the schema and info in a json string:

```
json_info = server.info.to_json()
json_schema = server.schema.to_json()
```

or can have them saved on file:

```
server.info.to_file('server-info.json)
server.schema.to_file('server-schema.json')
```

to build a new server object with the saved json files you can retrieve them with:

```
from ldap3 import DsaInfo, SchemaInfo
dsa_info = DsaInfo.from_file('server-info.json')
schema_info = SchemaInfo.from_file('server-schema.json')
server = Server('hostname', dsa_info, schema_info)
```

and then you can use the server as usual. Hostname must resolve to a real server.

Attribute missing

## Schema

An LDAP server store information about *types* it can handle in its **schema**. The schema includes all information needed by a client to correctly performs LDAP operations. Let's examine an LDAP server schema:

```
>>> server.schema
DSA Schema from: cn=schema
  Attribute types:{'ipaNTTrustForestTrustInfo': Attribute type: 2.16.840.1.113730.3.8.
↪11.17
  Short name: ipaNTTrustForestTrustInfo
  Description: Forest trust information for a trusted domain object
  Equality rule: octetStringMatch
  Syntax: 1.3.6.1.4.1.1466.115.121.1.40 [('1.3.6.1.4.1.1466.115.121.1.40', 'LDAP_
↪SYNTAX', 'Octet String', 'RFC4517')]
  'ntUserCreateNewAccount': Attribute type: 2.16.840.1.113730.3.1.42
  Short name: ntUserCreateNewAccount
  Description: Netscape defined attribute type
  Single Value: True
  Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
  Extensions:
    X-ORIGIN: Netscape NT Synchronization
  'passwordGraceUserTime': Attribute type: 2.16.840.1.113730.3.1.998
  Short name: passwordGraceUserTime, pwdGraceUserTime
  Description: Netscape defined password policy attribute type
  Single Value: True
  Usage: Directory operation
  Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
  Extensions:
    X-ORIGIN: Netscape Directory Server
  'nsslapd-ldapilisten': Attribute type: 2.16.840.1.113730.3.1.2229
  Short name: nsslapd-ldapilisten
  Description: Netscape defined attribute type
  Single Value: True
  Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
  Extensions:
    X-ORIGIN: Netscape Directory Server
  'bootParameter': Attribute type: 1.3.6.1.1.1.1.23
  Short name: bootParameter
  Description: Standard LDAP attribute type
  Syntax: 1.3.6.1.4.1.1466.115.121.1.26 [('1.3.6.1.4.1.1466.115.121.1.26', 'LDAP_
↪SYNTAX', 'IA5 String', 'RFC4517')]
  Extensions:
    X-ORIGIN: RFC 2307

  <...long list of descriptors...>
```

The schema is a very long list that describes what kind of data types the LDAP server understands. It also specifies what attributes can be stored in each class. Some classes are containers for other entries (either container or leaf) and are used to build the hierarchy of the DIT. Container entries can have attributes too. One important specification in the schema is if the attribute is *multi-valued* or not. A multi-valued attribute can store one or more values. Every LDAP server must at least support the standard LDAP3 schema but can have additional custom classes and attributes. The schema defines also the *syntaxes* and the *matching rules* of the different kind of data types stored in the LDAP.

**Note:** Object classes and attributes are independent objects. An attribute is not a "child" of a class neither a class is a

"parent" of any attribute. Classes and attributes are linked in the schema with the `MAY` and `MUST` options of the object class definition that specify what attributes an entry can contain and which of them are mandatory.

**Note:** There are 3 different types of object classes: **ABSTRACT** (used only when defining the class hierarchy), **STRUCTURAL** (used to create concrete entries) and **AUXILIARY** (used to add additional attributes to an entry). Only one structural class can be used in an entry, while many auxiliary classes can be added to the same entry. Adding an object class to an entry simply means that the attributes defined in that object class can be stored in that entry.

If the ldap3 library is aware of the schema used by the LDAP server it will try to automatically convert data retrieved by the Search operation to their representation. An integer will be returned as an int, a generalizedDate as a datetime object and so on. If you don't read the schema all the values are returned as bytes and unicode strings. You can control this behaviour with the `get_info` parameter of the Server object and the `check_names` parameter of the Connection object.

The schema can be extended by the user, but the LDAP RFCs don't specify how this operation must be performed, so each LDAP server has its own method of adding classes and attributes to the schema.

## Operational attributes

The LDAP server store *operational* information on each entry. This information is used by the internal mechanism of the server and *can* be made available to the user via an **operational attribute** that can usually be read but not written.

To request all operational attribute in a search you can use the + (PLUS) character as an attribute name. Keep in mind that the server may not return some operational attribute if they are not explicitly requested (because they may take a long time or many resources to be computed), so if you need a specific attribute is better to request it explicitly.

Some server may not return attribute information in the schema. In this case the ldap3 library is not aware of them. This can lead to some erratic behaviour, especially in the Abstraction Layer of ldap3.

In this case you can tell ldap3 to not check for a specific attribute:

```
from ldap3 import get_config_parameter, set_config_parameter
attrs = get_config_parameter('ATTRIBUTES_EXCLUDED_FROM_CHECK')
attrs.extend(['memberOf', 'entryUUID', 'pwdChangedTime'])  # # all the missing␣
↪attributes you need
set_config_parameter('ATTRIBUTES_EXCLUDED_FROM_CHECK', attrs)
```

Now the missing attributes can be used in searches.

Then, if you're using the Abstraction Layer you must instruct the ObjectDef to query for those attributes too. For example, if you want to query *inetOrgPerson* in a Reader Cursor of the Abstraction Layer:

```
from ldap3 import Connection, ObjectDef, Reader
c = Connection('my_server', 'my_user', 'my_password')
c.bind()
person = ObjectDef('inetOrgPerson', c)  # read the object class hierarchy schema from␣
↪the server
person += ['memberOf', 'entryUUID', 'pwdChangedTime'] # this creates the missing␣
↪AttrDef in the ObjectDef
r = Reader(c, person, 'my_base')
r.serch()
```

when you query the `r` cursor you'll get back the missing attributes too.

## Connection

The Connection object is used to send operation requests to the LDAP Server. It can use different connection strategies and supports the *context manager* protocol to automatically open, bind and unbind the connection.

The following strategies are available:

- SYNC: the request is sent and the connection waits until the response is received. You get the result in the return value of the connection.
- ASYNC: the request is sent and the connection immediately returns a *message_id* that can be used later to retrieve the response.
- LDIF: the request is transformed in a *ldif-change* format and an LDIF output is returned.
- RESTARTABLE: an automatically restartable synchronous connection. It retries operation for the specified number of times or forever.

---

**Note:** Lazy connections

In a lazy connection when you open() and bind() nothing is executed. These operation are deferred until an effective LDAP operation (add, modify, delete, compare, modifyDn, search, extended) is performed. If unbind() is executed when still in deferred status all deferred operation are cancelled and nothing is sent over the network. This can be helpful when your application opens connections ahead of knowing if an effective operation is needed.

---

- REUSABLE: an asynchronous strategy that internally opens multiple connections to the Server (or multiple Servers via the ServerPool) each in a different thread

When using an asynchronous strategy each operation returns immediately a message_id. You can call the get_response method of the connection object to obtain the response received from the server.

Connection parameters are:

- server: the Server object to be contacted. It can be a ServerPool. In this case the ServerPool pooling strategy is followed when opening the connection. You can also pass a string containing the name of the server. In this case the Server object is implicitly created with default values.
- user: the account of the user to log in for simple bind (defaults to None).
- password: the password of the user for simple bind (defaults to None)
- auto_bind: automatically opens and binds the connection. Can be AUTO_BIND_NONE, AUTO_BIND_NO_TLS, AUTO_BIND_TLS_AFTER_BIND, AUTO_BIND_TLS_BEFORE_BIND.
- version: LDAP protocol version (defaults to 3).
- authentication: authentication method, can be one of ANONYMOUS, SIMPLE, SASL or NTLM. Defaults to AUTH_ANONYMOUS if user and password are both None else defaults to AUTH_SIMPLE. NTLM uses NTLMv2 authentication. Username must be in the form domain\user.
- client_strategy: communication strategy used by the client (defaults to SYNC).
- auto_referrals: specify if the Connection must follows referrals automatically (defaults to True). Allowed referral servers are specified in the Server object.
- sasl_mechanism: specify the SASL mechanism to use for AUTH_SASL authentication. Available mechanism are EXTERNAL, DIGEST-MD5 (**deprecated** by RFCs because insecure) and GSSAPI.
- sasl_credential: an object specific to the SASL mechanism chosen. Refer to the documentation for each SASL mechanism supported.

- collect_usage: binds a ConnectionUsage object to the connection to store metrics of connection usage (see later).
- read_only: when True inhibits modify, delete, add and modifyDn (move) operations, defaults to False.
- lazy: when True connection will defer open and bind until another LDAP operation is requested
- check_names: when True attribute names in assertions and filters will be checked against the schema (Server must have schema loaded with the get_info=ALL or get_info=SCHEMA parameter) and search result will be formatted as specified in schema.
- raise_exceptions: when True LDAP operations will raise exceptions (subclasses of LDAPOperationResult) when the result is not one of the following: RESULT_SUCCESS, RESULT_COMPARE_FALSE, RESULT_COMPARE_TRUE, RESULT_REFERRAL.
- pool_name: an optional identifier for the Connection pool when using a pooled connection strategy
- pool_size: size of the connection pool used in a pooled connection strategy
- pool_lifetime: number of seconds before recreating a new connection in a pooled connection strategy
- pool_keepalive: number of seconds to wait before sending an Abandon(0) operation in an idle connection in a pooled connection strategy. Abandon(0) is an harmless LDAP operation used to not let the server closing the connection
- fast_decoder: when False use the pyasn1 decoder instead of the faster internal decoder. Gives a better output in extended log
- receive_timeout: set the socket in non-blocking mode - raising an exception after the specified amount of seconds if nothing is received over the wire
- return_empty_attributes: when a search is performed if an attribute is empty then sets its value to an empty list, default to True
- auto_range: if a server returns a fixed amount of entries in searches using the *range* tag (RFCs 3866) setting this value to True let the ldap3 library automatically request all entries with additional searches. The entries are returned as if a single search is performed
- use_referral_cache: when True referral connections are not immediately closed, and kept in a cache should another request need to contact the same server
- auto_escape: automatically applies LDAP encoding to filter values, default to True
- auto_encode: automatically tries to convert from local encoding to UTF8 for well known syntaxes and types, default to True

---

**Note:** The *auto_range* feature is very useful when searching Active Directory servers. When an Active Directory search returns more than 1000 entries this feature is automatically used by the server. So it can happens that your code works seamlessy until your data grow to exceed the 1000 entries limit and your code stops working properly without any apparent reason.

---

With the connection object you can perform all the standard LDAP operations:

- bind: performs a bind to the LDAP Server with the authentication type and credential specified in the connection:
  - controls: additional controls to send in the request
- unbind: disconnect and close the connection:
  - controls: additional controls to send in the request
- compare: performs a comparison between an attribute value of an entry and an arbitrary value:

  - dn: distinguished name of the entry whose attribute is to compare
  - attribute: name of the attribute to compare
  - value: value to be compared
  - controls: additional controls to send in the request
- add: add an entry to the LDAP server
  - dn: distinguished name of the object to add
  - object_class: class name of the attribute to add, can be a string containing a single value or a list of strings
  - attributes: a dictionary in the form {'attr1': 'val1', 'attr2': 'val2', ...} (or {'attr1': ['val1', 'val2', ...], ...} for multivalued attributes)
  - controls: additional controls to send in the request
- delete: deletes the object specified:
  - dn: distinguished name of the object to delete
  - controls: additional controls to send in the request
- modify: modifies attributes of an entry:
  - dn: distinguished name of the object whose attributes must be modified
  - changes: a dictionary in the form {'attribute1': [(operation1, [val1, val2, ...]), (operation2, [val1, val2, ...]), ...]}, operation is MODIFY_ADD, MODIFY_DELETE, MODIFY_REPLACE, MODIFY_INCREMENT
  - controls: additional controls to send in the request
- modify_dn: modifies the relative distinguished name of an entry or performs a move of an entry:
  - dn: distinguished name of the entry whose relative name must be modified
  - relative_dn: new relative dn of the entry
  - delete_old_dn: remove the previous dn (defaults to True)
  - new_superior: the new container of the entry
  - controls: additional controls to send in the request

---

**Note:** modify_dn is really a two-flavours operation: you can rename the last part of the dn *or* you move the entry in another container but you cannot perform both operations at the same time.

---

- Search: performs a search in the LDAP database:
  - search_base: the base of the search request.
  - search_filter: the filter of the search request. It must conform to the LDAP filter syntax specified in RFC4515. If the search filter contains the following characters you must use the relevant escape ASCII sequence, as per RFC4515 (section 3): '*' -> '\\2A', '(' -> '\\28', ')' -> '\\29', '\' -> '\\5C', chr(0) -> '\\00'.
  - search_scope: specifies how broad the search context is:
    - BASE: retrieves attributes of the entry specified in the search_base.
    - LEVEL: retrieves attributes of the entries specified in the search_base. The base must reference a container object.
    - SUBTREE: retrieves attributes of the entries specified in the search_base and all subordinate containers downward.

  - dereference_aliases: specifies how the server must treat references to other entries:
    - DEREF_NEVER: never dereferences entries, returns alias objects instead. The alias contains the reference to the real entry.
    - DEREF_SEARCH: while searching subordinates of the base object, dereferences any alias within the search scope. Dereferenced objects become the bases of further search scopes where the Search operation is also applied. The server should eliminate duplicate entries that arise due to alias dereferencing while searching.
    - DEREF_BASE: dereferences aliases in locating the base object of the search, but not when searching subordinates of the base object.
    - DEREF_ALWAYS: always returns the referenced entries, not the alias object.
  - attributes: a single attribute or a list of attributes to be returned by the search (defaults to None). If attributes is None no attribute is returned. If attributes=ALL_ATTRIBUTES all attributes are returned, if attributes=ALL_OPERATIONAL_ATTRIBUTES all operational attributes are returned. To get both use attributes=[ALL_ATTRIBUTES, ALL_OPERATIONAL_ATTRIBUTES].
  - size_limit: maximum number of entries returned by the search (defaults to None). If None the whole set of found entries is returned, unless the server has a more restrictive constrai.
  - time_limit: number of seconds allowed for the search (defaults to None). If None the search can take an unlimited amount of time, unless the server has a more restrictive constrain.
  - types_only: doesn't return attribute values.
  - get_operational_attributes: if True returns information attributes (managed automatically by the server) for each entry.
  - controls: additional controls to send in the request.
  - paged_size: if paged_size is greater than 0 a simple paged search is executed as described in RFC2696 (defaults to None). The search will return at most the specified number of entries.
  - paged_criticality: if True the search will be executed only if the server is capable of performing a simple paged search. If False and the server is not capable of performing a simple paged search a standard search will be executed.
  - paged_cookie: an *opaque* string received in a paged paged search that must be sent back while requesting subsequent entries of the search result.

- Abandon: abandons the operation indicated by message_id, if possible:
  - message_id: id of a previously sent request
  - controls: additional controls to send in the request to be abandoned
- Extended: performs an extended operation:
  - request_name: name of the extended operation
  - request_value: optional value sent in the request (defaults to None)
  - controls: additional controls to send in the request
  - no_encode: when True the value is passed without any encoding (defaults to False)

Additional methods defined:

- start_tls: establishes a secure connection, can be executed before or after the bind operation.
- do_sasl_bind: performs a SASL bind with the parameter defined in the Connection. It's automatically executed when you call the bind operation if SASL authentication is used.

- refresh_dsa_info: reads info from server as specified in the get_info parameter of the Connection object.
- response_to_ldif: a method you can call to convert the response of a search to a LDIF format (ldif-content). It has the following parameters:
  - search_result: the result of the search to be converted (defaults to None). If None get the last response received from the Server
  - all_base64: converts all the value to base64 (defaults to False)
- response_to_json: this method returns the entries found in a search in a string with JSON format
- response_to_file: this method saves to a file the entries found in a search with JSON format. You can specify if you want the raw attributes with the raw=True parameter. Entries are saved as a list in the 'entries' key.

Connection attributes:

- server: the active Server object used in the connection
- server_pool: the ServerPool object used in the connection if available
- read_only: True if the connection is in read only mode
- version: the LDAP protocol version used
- result: the result of the last operation
- response: the response of the last operation (for example, the entries found in a search), without the result
- last_error: any error occurred in the last operation (for synchronous strategies)
- bound: True if bound to server else False
- listening: True if the socket is listening to the server
- closed: True if the socket is not open
- strategy_type: the strategy type used by the connection
- strategy: the strategy instance used by the connection
- authentication: the authentication type used in the connection
- user: the user name for simple bind
- password: password for simple bind
- auto_bind: True if auto_bind is active else False
- tls_started: True if the Transport Security Layer is active
- usage: metrics of connection usage
- lazy: connection will defer open and bind until another LDAP operation is requested
- check_names: True if you want to check the attribute and object class names against the schema in filters and in add/compare/modify operations
- pool_name: an optional identifier for the Connection pool when using a pooled connection strategy
- pool_size: size of the connection pool used in a pooled connection strategy
- pool_lifetime: number of seconds before recreating a new connection in a pooled connection strategy

## Controls

Controls, if used, must be a list of tuples. Each tuple must have 3 elements: the control OID, a boolean to specify if the control is critical, and a value. If the boolean is set to True the server must honorate the control or refuse the operation. Mixing controls must be defined in controls specification (as per RFC4511). controlValue is optional, set it to None to not send any value.

## Result

Each operation has a result stored as a dictionary in the connection.result attribute. You can check the result value to know if the operation has been sucessful. The dictionary has the following field:

- result: the numeric result code of the operation as specified in RFC4511
- description: extended description of the result code, as specified in RFC4511
- message: a diagnostic message sent by the server (optional)
- dn: a distinguish name of an entry related to the request (optional)
- referrals: a list of referrals where the operation can be continued (optional)

## Responses

Responses are received and stored in the connection.response as a list of dictionaries. You can get the search result entries of a Search operation iterating over the response attribute. Each entry is a dictionary with the following field:

- dn: the distinguished name of the entry
- attributes: a dictionary of returned attributes and their values. Values are in UTF-8 format. If the Connection is aware of the server schema, values are properly stored: directly for single-valued attributes and as a list for multi-valued attributes. A multi-valued attribute with a single value is always stored as a list. If the server schema is unkwown all values are stored as a list.
- raw_attributes: the unencoded values, always stores as a list of bytearray regardless of the schema definition.

## Checked Attributes

The checked attributes feature checks the LDAP syntax of the attributes defined in schema and returns a properly formatted entry value while performing searches. This means that if, for example, you have an attributes specified as GUID in the server schema you will get the properly formatted GUID value ('012381d3-3b1c-904f-b29a-012381d33b1c') in the connection.response[0]['attributes'] key dictionary instead of a sequence of bytes. Or if you request an attribute defined as an Integer in the schema you will get the value already converted to int. Furthermore for attributes defined *single valued* in the schema you will get the value instead of a list containing only one value. To activate this feature you must set the get_info parameter to SCHEMA or ALL when defining the server object and the check_names attributes to True in the Connection object (the default).

There are some standard formatters defined in the library, most of them are defined in the relevants RFCs:

- format_unicode # returns an unicode object in Python 2 and a string in Python 3
- format_integer # returns an integer
- format_binary # returns a bytes() sequence
- format_uuid # returns a GUID (UUID) as specified in RFC 4122 - byte order is big endian
- format_uuid_le # same as above but byte order is little endian

- format_boolean # returns a boolean
- format_time # returns a datetime object (with properly defined timezone, or UTC if timezone is not specified) as defined in RFC 4517

You can even define your custom formatter for specific purposes. Just pass a dictionary in the format {'identifier': callable} in the 'formatter' parameter of the Server object. The callable must be able to receive a bytes value and convert it to the relevant object or class instance.

Custom formatters have precedence over standard formatter. In each category (from highest to lowest) the resolution order is:

1. attribute name
2. attribute oid (from schema)
3. attribute names (from oid_info)
4. attribute syntax (from schema)

If a suitable formatter is not found the value will be rendered as bytes.

# SSL and TLS

You can use SSL basic authentication with the use_ssl parameter of the Server object, you can also specify a port (636 is the default for secure ldap):

```
s = Server('servername', port = 636, use_ssl = True)  # define a secure LDAP server
```

To start a TLS connection on an already created _clear connection:

```
c.start_tls()
```

Some older versions (up to 2.7.9) of the Python interpreter lack the capability to check the server certificate against the DNS name of the server. This is a potential breach of security because a server could present a certificate issued for another host name. ldap3 includes a backport of this capability ported from the 3.4.3 version of the Python interpreter. If you want to keep your application up to date with the hostname checking capability of the latest Python version you can install the backports.ssl_match_hostname package from pypi. The ldap3 library will detect and use it instead of the included static backport.

## The Tls object

You can customize the server Tls object with references to keys, certificates and CAs. It includes all attributes needed to securely connect over an ssl socket:

- local_private_key_file: the file with the private key of the client
- local_certificate_file: the certificate of the server
- validate: specifies if the server certificate must be validated, values can be: CERT_NONE (certificates are ignored), CERT_OPTIONAL (not required, but validated if provided) and CERT_REQUIRED (required and validated)
- version: SSL or TLS version to use, can be one of the following: SSLv2, SSLv3, SSLv23, TLSv1 (as per Python 3.3. The version list can be different in other Python versions)
- ca_certs_file: the file containing the certificates of the certification authorities

- ciphers: a string that specify which chipers must be used. It works on recent Python interpreters that allow to change the cipher in the SSLContext or in the the wrap_socket() method, it's ignored on older versions.

Tls object uses the ssl module of the Python standard library with additional checking functions that are missing from the Python 2 standard library.

The needed constants are defined in the ssl package.

IF you don't use a specific Tls object and set use_ssl=True in the Server definition, a default Tls object will be used, it has no certificate files, uses the ssl.PROTOCOL_SSLv23 (if available in your Python interpreter) and performs no validation of the server certificate. It's recommended to set validate=ssl.CERT_REQUIRED to verify the certificate server. Example:

```
tls = Tls(local_private_key_file='client_private_key.pem', local_certificate_file=
↪'client_cert.pem', validate=ssl.CERT_REQUIRED, version=ssl.PROTOCOL_TLSv1, ca_certs_
↪file='ca_certs.b64')
```

### SSLContext

You can use SSLContext if running in Python 3.4 or newer.

The use of ssl.SSLContext make TLS operation more flexible, It integrates with the system wide Certification Authorities and also ensure that there are "reasonable" security defaults when using the TLS layer. It's also possible to specify a file system path containing the CA file or even pass certificate data "on the fly".

When defining the Tls object you have the following additional parameters available:

- ca_cert_file: the usual link to the certification authority chain of certificates
- ca_cert_path: a link to a path containing the certification authorities certificates (reashed, as expected by OpenSSL)
- ca_cert_data: CA certificate data stored in memory

if you leave all these parameter to None the SSLContext will use the system wide certificate store (ssl path on linux, CA stores on Windows)

If the SSLContext is not available the library will fall back to the ssl wrapped socket mechanism.

### SASL

Three SASL mechanisms are currently implemented in the ldap3 library: EXTERNAL, DIGEST-MD5 and GSSAPI (Kerberos, via the gssapi package). Even if DIGEST-MD5 is **deprecated** and moved to historic (RFC6331, July 2011) because it is **insecure and unsuitable for use in protocols** (as stated by the RFC) I've developed the authentication phase of the protocol because it is still used in LDAP servers.

### External

You can use the EXTERNAL mechanism when you're on a secure (TLS) channel. You can provide an authorization identity string in sasl_credentials or let the server trust the credential provided when establishing the secure channel:

```
tls = Tls(local_private_key_file = 'key.pem', local_certificate_file = 'cert.pem',␣
↪validate = ssl.CERT_REQUIRED, version = ssl.PROTOCOL_TLSv1,
          ca_certs_file = 'cacert.b64')
server = Server(host = test_server, port = test_port_ssl, use_ssl = True, tls = tls)
```

```
connection = Connection(server, auto_bind = True, version = 3, client_strategy = test_
↪strategy, authentication = SASL,
                        sasl_mechanism = 'EXTERNAL', sasl_credentials = 'username')
```

### Digest-MD5

To use the DIGEST-MD5 you must pass a 4-value tuple as sasl_credentials: (realm, user, password, authz_id). You can pass None for 'realm' and 'authz_id' if not used. Quality of Protection is always 'auth':

```
server = Server(host = test_server, port = test_port)
connection = Connection(server, auto_bind = True, version = 3, client_strategy = test_
↪strategy, authentication = SASL,
                        sasl_mechanism = 'DIGEST-MD5', sasl_credentials = (None,
↪'username', 'password', None))
```

Username is not required to be an LDAP entry, but it can be any identifier recognized by the server (i.e. email, principal, ...). If you pass None as 'realm' the default realm of the LDAP server will be used.

**Again, remember that DIGEST-MD5 is deprecated and should not be used.**

## Connection metrics

If you set the collect_usage parameter to True in a Connection object the metrics feature is activated. You get the 'usage' attribute in the connection object populated with an instance of the ConnectionUsage class.

ConnectionUsage stores counters for each operation performed in the Connection, you get metrics for the following fields:

- initial_connection_start_time
- open_socket_start_time:
- connection_stop_time:
- servers_from_pool:
- open_sockets:
- closed_sockets:
- wrapped_sockets:
- bytes_transmitted:
- bytes_received:
- messages_transmitted:
- messages_received:
- operations:
- abandon_operations:
- bind_operations:
- add_operations:
- compare_operations:
- delete_operations:

- extended_operations:
- modify_operations:
- modify_dn_operations:
- search_operations:
- unbind_operations:
- referrals_received:
- referrals_followed:
- referrals_connections:
- restartable_failures:
- restartable_successes:

Metrics are properly collected while the connection is open, kept while it's closed and reset if the connection is used again. While using a ServerPool or a restartable strategy the metrics are not reset when the server is changed.

You can reset the usage metrics with the connection.usage.reset() method.

You can print out the metrics at any time of execution of your code with:

```
print(connection.usage)
```

and get the formatted metrics:

```
Connection Usage:
  Time: [elapsed:         0:00:00.028885]
    Initial start time:  2015-06-29T01:44:17.772645
    Open socket time:    2015-06-29T01:44:17.772645
    Close socket time:   2015-06-29T01:44:17.801530
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 62
    Transmitted:         48
    Received:            14
  Messages:              3
    Transmitted:         2
    Received:            1
  Operations:            2
    Abandon:             0
    Bind:                1
    Add:                 0
    Compare:             0
    Delete:              0
    Extended:            0
    Modify:              0
    ModifyDn:            0
    Search:              0
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
    Connections:         0
  Restartable tries:     0
```

```
    Failed restarts:     0
    Successful restarts: 0
```

# LDAP Operations

## The BIND operation

As specified in RFC4511 the **Bind** operation is the "authenticate" operation. It (and the Unbind operation as well) has this name for historical reason.

When you open a connection to an LDAP server you're in an **anonymous** connection state. What this exactly means is defined by the server implementation, not by the protocol. Think of this as a public access to the server data (even if what public data mean is still a server matter). In ldap3 you establish the connection to the server with the `open()` method of the Connection object. The `bind()` method will open the connection if not already open.

The Bind operation allows creadentials to be exchanged between the client and server to establish a new authorization state.

The Bind request typically specifies the desired authentication identity. Some Bind mechanisms also allow the client to specify the authorization identity. If the authorization identity is not specified, the server derives it from the authentication identity in an implementation-specific manner.

If you want to provide authentication information you must use the Bind operation to specify an identity to use to access the data. Keep in mind that either the authentication details than the authorization details are a local server matter. The LDAP protocol doesn't specify how the identity must be stored on the server nor how the authorization ACLs are specified.

The Bind operation specify 4 different methods to authenticate to the server, as specified in RFC4513:

- Simple Bind: you provide user credentials by the means of a username (in a dn form) and a password.
- Anonymous Bind: the user and password are passed as empty strings.
- Unauthenticated simple Bind: you pass a username without a password. This method, even if specified in the protocol, should not be used because is higly insecure and should be forbidden by the server. It was used in the past for tracing purpose.
- SASL (Simple Authentication and Security Layer): this defines multiple mechanisms that each server can provide to allow access to the server. Before trying a mechanism you should check that the server supports it. The LDAP server publish its allowed SASL mechanism in the DSE information that can be read anonymously with the `get_info=ALL` parameter of the Server object.

The Bind method returns True if the bind is successful, False if something goes wrong while binding. In this case you can inspect the `result` attribute of the Connection object to get the error description.

### Simple Bind

You perform a Simple Bind operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema
```

```
# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the Bind operation
if not c.bind():
    print('error in bind', c.result)
```

The server and the connection are created with the default parameters:

```
s = Server(host='servername', port=389, use_ssl=False, get_info='ALL')
c = Connection(s, user='user_dn', password='user_password', auto_bind='NONE',␣
↪version=3, authentication='SIMPLE', \
client_strategy='SYNC', auto_referrals=True, check_names=True, read_only=False,␣
↪lazy=False, raise_exceptions=False)
```

Refer to the Server and Connection docs for info about the default parameters.

### Anonymous Bind

Anonymous bind performs a simple bind with the user name and the user password set to empty strings. The ldap3 library has a specific authentication option to do that:

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s)  # define an ANONYMOUS connection

# perform the Bind operation
if not c.bind():
    print('error in bind', c.result)
```

The server and the connection are created with the default parameters:

```
s = Server(host='servername', port=389, use_ssl=False, get_info='ALL')
c = Connection(s, auto_bind='NONE', version=3, authentication='ANONYMOUS', client_
↪strategy='SYNC', auto_referrals=True, \
check_names=True, read_only=False, lazy=False, raise_exceptions=False)
```

To use SSL basic authentication change the server definition to:

```
s = Server('servername', use_ssl=True, get_info=ALL)  # define a secure LDAP server␣
↪on the default 636 port
```

### StartTLS

If you want to raise the transport layer security to an encrypted state you can perform the *StartTLS* extended operation. With this mechanism you can wrap the plain socket in an SSL encrypted socket:

```
c.start_tls()
```

From now on the communication transport is encrypted. You should properly configure the Server object addind a Tls object with the relevant ssl configuration:

```
t = Tls(local_private_key_file='client_private_key.pem', local_certificate_file=
↪'client_cert.pem', validate=ssl.CERT_REQUIRED, version=ssl.PROTOCOL_TLSv1, ca_certs_
↪file='ca_certs.b64')
s = Server('servername', tls=t, get_info=ALL)
```

Please refer to the SSLTLS section for more information.

### SASL

Three SASL mechanisms are currently implemented in the ldap3 library: EXTERNAL, DIGEST-MD5, GSSAPI (Kerberos, via the gssapi package) and PLAIN. DIGEST-MD5 is implemented even if it is **deprecated** and moved to historic (RFC6331, July 2011) because it is **"insecure and unsuitable for use in protocols"** (as stated by the RFC).

To query the SASL mechanism available on the server you must read the information published by the server. The ldap3 library has a convenient way to do that:

```
from ldap3 import Server, Connection, ALL
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema
c = Connection(s)
c.open()  # establish connection without performing any bind (equivalent to ANONYMOUS␣
↪bind)
print(s.info.supported_sasl_mechanisms)
```

Print out a list of the SASL mechanism supported by the server:

```
['EXTERNAL', 'DIGEST-MD5', 'GSSAPI']
```

### External

You can use the EXTERNAL mechanism when you're on a secure (TLS) channel. You can provide an authorization identity string in `sasl_credentials` or let the server trust the credential provided when establishing the secure channel:

```
from ldap3 import Server, Connection, Tls, SASL,EXTERNAL
tls = Tls(local_private_key_file = 'key.pem', local_certificate_file = 'cert.pem',␣
↪validate = ssl.CERT_REQUIRED, version = ssl.PROTOCOL_TLSv1,
          ca_certs_file = 'cacert.b64')
server = Server(host = test_server, port = test_port_ssl, use_ssl = True, tls = tls)
c = Connection(server, auto_bind = True, version = 3, client_strategy = test_strategy,
↪ authentication = SASL,
                        sasl_mechanism = EXTERNAL, sasl_credentials = 'username')
```

### Digest-MD5

To use the DIGEST-MD5 mechanism you must pass a 4-value tuple as sasl_credentials: (realm, user, password, authz_id). You can pass None for 'realm' and 'authz_id' if not used. Quality of Protection is always 'auth':

```
from ldap3 import Server, Connection, SASL, DIGEST_MD5
server = Server(host = test_server, port = test_port)
```

```
c = Connection(server, auto_bind = True, version = 3, client_strategy = test_strategy,
↪ authentication = SASL,
                          sasl_mechanism = DIGEST_MD5, sasl_credentials = (None,
↪'username', 'password', None))
```

Username is not required to be an LDAP entry, but it can be any identifier recognized by the server (i.e. email, principal, ...). If you pass None as 'realm' the default realm of the LDAP server will be used.

**Again, remember that DIGEST-MD5 is deprecated and should not be used.**

### Kerberos

Kerberos authentication uses the `gssapi` package. You must install it and configure your Kerberos environment to use the GSSAPI mechanism:

```
from ldap3 import Server, Connection, Tls, SASL, KERBEROS
import ssl
tls = Tls(validate=ssl.CERT_NONE, version=ssl.PROTOCOL_TLSv1_2)
server = Server('<servername>', use_ssl=True, tls=tls)
c = Connection(
    server, authentication=ldap3.SASL, sasl_mechanism=KERBEROS)
c.bind()
print(c.extend.standard.who_am_i())
```

You can specify which Kerberos client principal should be used with the `user` parameter when declaring the Connection:

```
c = Connection(
    server, user='ldap-client/client.example.com',
    authentication=SASL, sasl_mechanism=KERBEROS)
```

By default the library attempts to bind against the service principal for the domain you attempted to connect to. If your target LDAP service uses a round-robin DNS, it's likely that the hostname you connect to won't match. In this case, you can either specify a hostname explicitly as the first element of the `sasl_credentials` connection parameter, or pass `True` as the first element to do a reverse DNS lookup:

```
# Override server hostname for authentication
c = Connection(
    server, sasl_credentials=('ldap-3.example.com',),
    authentication=SASL, sasl_mechanism=KERBEROS)

# Perform a reverse DNS lookup to determine the hostname to authenticate against.
c = Connection(server, sasl_credentials=(True,), authentication=SASL, sasl_
↪mechanism=KERBEROS)
```

### Plain

The PLAIN SASL mechanism sends data in clear text, so it must rely on other means of securing the connection between the client and the LDAP server. As stated in RFC4616 the PLAIN mechanism should not be used without adequate data security protection as this mechanism affords no integrity or confidentiality protections itself. The mechanism is intended to be used with data security protections provided by application-layer protocol, generally through its use of Transport Layer Security (TLS) services.

To use the PLAIN mechanism you must pass a 3-value tuple as sasl_credentials: (authorization_id, authentication_id, password). You can pass None for authorization_id if it is not used:

```
from ldap3 import Server, Connection, SASL, PLAIN
server = Server(host = test_server, port = test_port, use_ssl=True)
c = Connection(server, auto_bind=True, authentication=SASL, sasl_mechanism=PLAIN,␣
↪sasl_credentials=(None, 'username', 'password'))
```

### NTLM

The ldap3 library supports an additional method to bind to Active Directory servers via the NTLM method:

```
# import class and constants
from ldap3 import Server, Connection, SIMPLE, SYNC, ALL, SASL, NTLM

# define the server and the connection
s = Server('servername', get_info=ALL)
c = Connection(s, user="AUTHTEST\\Administrator", password="password",␣
↪authentication=NTLM)
# perform the Bind operation
if not c.bind():
    print('error in bind', c.result)
```

This authentication method is specific for Active Directory and uses a proprietary authentication protocol named SICILY that breaks the LDAP RFC but can be used to access AD.

When binding via NTLM, it is also possible to authenticate with an LM:NTLM hash rather than a password:

```
c = Connection(s, user="AUTHTEST\\Administrator", password=
↪"E52CAC67419A9A224A3B108F3FA6CB6D:8846F7EAEE8FB117AD06BDD830B7586C",␣
↪authentication=NTLM)
```

### LDAPI (LDAP over IPC)

If your LDAP server provides a UNIX socket connection you can use the **ldapi:** (*Interprocess Communication*) scheme to access it from the same machine:

```
>>> # accessing OpenLDAP server in a root user session
>>> s = Server('ldapi:///var/run/slapd/ldapi')
>>> c = Connection(s, authentication=SASL, sasl_mechanism=EXTERNAL, sasl_credentials='
↪')
>>> c.bind()
>>> True
>>> c.extend.standard.who_am_i()
>>> dn:cn=config
```

Using the SASL *EXTERNAL* mechanism allows you to provide to the server the credentials of the logged user.

While accessing your LDAP server via a UNIX socket you can perform any usual LDAP operation. This should be faster than using a TCP connection. You don't need to use SSL when connecting via a socket because all the communication is in the server memory and is not exposed on the wire.

### Bind as a different user while the Connection is open

LDAP protocol allows to bind as a different user while the connection is open. In this case you can use the **rebind()** method that let you change the user and the authentication method while the connection is open:

```
# import class and constants
from ldap3 import Server, Connection, ALL, LDAPBindError

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the Bind operation
if not c.bind():
    print('error in bind', c.result)

# Bind again with another user
if not c.rebind(user='different_user_dn', password='different_user_password')
    print('error in rebind', c.result)
```

In case the credentials are invalid or if the server doesn't allow you to rebind the server *could* abruptly close the connection. This condition is checked by the `rebind()` method and an LDAPBindError exception will be raised if caught.

If you want an exception raised when credentials are invalid you must use the `raise_exceptions=True` parameter in the Connection() definition. Keep in mind that network errors always raise an exception, even if raise_exceptions is set to False.

### Extended logging

To get an idea of what's happening when you perform a Simple Bind operation using the StartTLS security feature this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=test', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
```

```
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:49610 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:start START TLS operation via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:instantiated Tls: <Tls(validate=0)>


# Starting TLS - wrapping the socket in an ssl socket:

DEBUG:ldap3:BASIC:starting tls for <ldap://openldap:389 - cleartext - user: cn=admin,
↪o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.137.104:389>
↪ - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:start EXTENDED operation via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:EXTENDED request <{'name': '1.3.6.1.4.1.1466.20037', 'value':␣
↪None}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -
↪ open - <local: 192.168.137.1:49611 - remote: 192.168.137.104:389> - tls not␣
↪started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  extendedReq=ExtendedRequest:
>>   requestName=b'1.3.6.1.4.1.1466.20037'
DEBUG:ldap3:NETWORK:sent 31 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  extendedResp=ExtendedResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:EXTENDED response <[{'referrals': None, 'dn': '', 'type':
↪'extendedResp', 'result': 0, 'description': 'success', 'responseName': None,
↪'responseValue': b'', 'message': ''}]> received via <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 -␣
↪remote: 192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done EXTENDED operation, result <True>
DEBUG:ldap3:BASIC:tls started for <ldap://openldap:389 - cleartext - user: cn=admin,
↪o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.137.104:389>
↪ - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:socket wrapped with SSL using SSLContext for <ldap://openldap:389␣
↪- cleartext - user: cn=admin,o=test - unbound - open - <local: [None]:None -␣
↪remote: [None]:None> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done START TLS operation, result <True>


# Performing Bind operation with Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'name': 'cn=admin,o=test', 'authentication
↪': {'sasl': None, 'simple': '<stripped 8 characters of sensitive data>'}, 'version
↪': 3}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -
↪ open - <local: 192.168.137.1:49611 - remote: 192.168.137.104:389> - tls started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=test'
>>   authentication=AuthenticationChoice:
```

```
>>    simple=b'<stripped 8 characters of sensitive data>'
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.
↪137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49611 - remote: 192.
↪168.137.104:389> - tls started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'referrals': None, 'dn': '', 'type':
↪'bindResponse', 'result': 0, 'description': 'success', 'saslCreds': None, 'message
↪': ''}> received via <ldap://openldap:389 - cleartext - user: cn=admin,o=test -␣
↪unbound - open - <local: 192.168.137.1:49611 - remote: 192.168.137.104:389> - tls␣
↪started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:49611 - remote: 192.168.137.
↪104:389> - tls started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
 Time: [elapsed:         0:00:01.908938]
   Initial start time:  2015-06-02T09:37:49.451263
   Open socket time:    2015-06-02T09:37:49.451263
   Close socket time:
 Server:
   Servers from pool:   0
   Sockets open:        1
   Sockets closed:      0
   Sockets wrapped:     1
 Bytes:                 96
   Transmitted:         68
   Received:            28
 Messages:              4
   Transmitted:         2
   Received:            2
 Operations:            2
   Abandon:             0
   Bind:                1
   Add:                 0
   Compare:             0
   Delete:              0
   Extended:            1
   Modify:              0
   ModifyDn:            0
   Search:              0
```

```
   Unbind:               0
 Referrals:
   Received:             0
   Followed:             0
 Restartable tries:      0
   Failed restarts:      0
   Successful restarts: 0
```

As you can see there have been two operation, one for the bind and one for the startTLS (an extendend operation). One socket has been open and has been wrapped in SSL. All the communication stream took 96 bytes in 4 LDAP messages.

## The UNBIND operation

As specified in RFC4511 the **Unbind** operation must be tought as the "disconnect" operation. It's name (and that of its Bind counterpart) is for historical reason.

Bind and Unbind are not symmetrical operations. In fact when you open a connection to an LDAP server you're already bounded in an **anonymous** connection state. What this exactly means is defined by the server implementation, not by the protocol. When you perform an Unbind operation you're actually requesting the server to end the user's sessione and close the communication socket. The Bind operation instead has nothing to do with the socket, but performs the user's authentication.

So it's perfectly legal to open a connession to the ldap server, perform some operation in the anonymous state and then Unbind to close the session.

In the ldap3 library the signature for the Delete operation is:

```
def unbind(self,
           controls=None):
```

- controls: additional controls to send in the request

The unbind method always returns True.

You perform an Unbind operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform some LDAP operation
...

# perform the Unbind operation
c.unbind()
```

The Unbind request is quite peculiar in the LDAPv3 protocol. There is no acknowledgement from thet server, that doesn't respond at all. It just ends the user's session and close the socket. The ldap3 library checks the success of this operation shutting down the socket on both communication directions and then closes the socket.

You can check if the socket has been closed querying the *closed* attribute of the connection object.

### Notice of Disconnection

Usually all communications between the client and the server are started by the client. There is only one case where the LDAP server send an unsolicited message: the Notice of Disconnection. This message is issued by the server as an alert when the server is about to stop. Once the message is sent, the server doesn't wait for any response, closes the socket and quits. Note that this notification is not used as a response to an Unbind requested by the client.

When the ldap3 library receive a Notice of Disconnection it tries to gracefully close the socket and then raise the LDAPSessionTerminatedByServer Exception. With asynchronous strategies the Exception is raised immediately, while with synchronous strategies the Exception is raised as you try to send data over the socket.

### Extended logging

To get an idea of what's happening when you perform an Unbind operation this is the extended log from an unbounded session to an OpenLdap server from a Windows client:

```
# Initialization:
INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=test', password=
↪'password', auto_bind='NONE', version=3, authentication='SIMPLE', client_strategy=
↪'SYNC', auto_referrals=True, check_names=True, collect_usage=True, read_only=False,␣
↪lazy=False, raise_exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:49610 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>


# Performing the Unbind operation:

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:49291 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:01.030738]
    Initial start time:  2015-06-04T17:01:43.431465
    Open socket time:    2015-06-04T17:01:43.431465
    Close socket time:   2015-06-04T17:01:44.462203
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 7
    Transmitted:         7
    Received:            0
  Messages:              1
    Transmitted:         1
```

```
    Received:            0
  Operations:            1
    Abandon:             0
    Bind:                0
    Add:                 0
    Compare:             0
    Delete:              0
    Extended:            0
    Modify:              0
    ModifyDn:            0
    Search:              0
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
  Restartable tries:     0
    Failed restarts:     0
    Successful restarts: 0
```

As you can see there is been one operation only, the Unbind operation. 1 socket has been open and then has been closed. All the communication stream took 7 bytes in 1 LDAP messages and the server never sent anything back.

## The ADD operation

The **Add** operation allows a client to request the addition of an entry into the LDAP directory. The Add operation is used only for new entries, that is the dn must reference a non existent object, but the parent objects must exist. For example if you try to add an entry with dn cn=user1,ou=users,o=company the company and users containers must already be present in the directory but the user1 object must non exist.

To perform an Add operation you must specify the dn of the new entry and a list of attributes to add.

In the ldap3 library the signature for the Add operation is:

```
def add(self,
        dn,
        object_class=None,
        attributes=None,
        controls=None)
```

- dn: distinguish name of the object to add
- object_class: class name of the attribute to add, can be a string containing a single value or a list of strings
- attributes: a dictionary in the form {'attr1': 'val1', 'attr2': 'val2', ...} or {'attr1': ['val1', 'val2', ...], ...} for multivalued attributes
- controls: additional controls to send with the request

For synchronous strategies the add method returns True if the operation was successful, returns False in case of errors. In this case you can inspect the result attribute of the connection object to get the error description.

For asynchronous strategies the add method returns the message id of the operation. You can get the operation result with the `get_response(message_id)` method of the connection object. If you use the `get_request=True` parameter you get the request dictionary back.

The object_class parameter is a shortcut for specify a sequence of object classes. You can specify the object classes in the attributes parameter too.

You perform an Add operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the Add operation
c.add('cn=user1,ou=users,o=company', ['inetOrgPerson', 'posixGroup', 'top'], {'sn':
↪'user_sn', 'gidNumber': 0})
# equivalent to
c.add('cn=user1,ou=users,o=company', attributes={'objectClass':  ['inetOrgPerson',
↪'posixGroup', 'top'], 'sn': 'user_sn', gidNumber: 0})

print(c.result)

# close the connection

c.unbind()
```

Obviously you must follow all the rules and restrictions specified in the LDAP server schema. Furthermore you cannot specify any operational attributes or any attribute defined with the NO-USER-MODIFICATION flag in the schema, or the operation will fail.

### Extended logging

To get an idea of what's happening when you perform an Add operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=test', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
```

```
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'authentication': {'simple': '<stripped 8␣
↪characters of sensitive data>', 'sasl': None}, 'name': 'cn=admin,o=test', 'version
↪': 3}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -
↪ open - <local: 192.168.137.1:50397 - remote: 192.168.137.104:389> - tls not␣
↪started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=test'
>>   authentication=AuthenticationChoice:
>>    simple=b'<stripped 8 characters of sensitive data>'
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.168.␣
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.␣
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50397 - remote: 192.␣
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'referrals': None, 'type': 'bindResponse',␣
↪'result': 0, 'message': '', 'dn': '', 'saslCreds': None, 'description': 'success'}>␣
↪received via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:50397 - remote: 192.168.137.104:389> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.␣
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>


# Performing the Add operation:

DEBUG:ldap3:BASIC:start ADD operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.␣
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:ADD request <{'entry': 'cn=user1,o=test', 'attributes': {␣
↪'gidNumber': ['0'], 'sn': ['user_sn'], 'objectClass': ['inetOrgPerson', 'posixGroup␣
↪', 'top']}}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test -␣
↪bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.104:389> - tls not␣
↪started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.␣
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.␣
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  addRequest=AddRequest:
>>   entry=b'cn=user1,o=test'
>>   attributes=AttributeList:
>>    Attribute:
>>     type=b'gidNumber'
>>     vals=ValsAtLeast1:
>>      b'0'
>>    Attribute:
>>     type=b'sn'
>>     vals=ValsAtLeast1:
>>      b'user_sn'
```

```
>>    Attribute:
>>     type=b'objectClass'
>>     vals=ValsAtLeast1:
>>      b'inetOrgPerson'      b'posixGroup'      b'top'
DEBUG:ldap3:NETWORK:sent 110 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  addResponse=AddResponse:
<<   resultCode='entryAlreadyExists'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:ADD response <[{'referrals': None, 'type': 'addResponse', 'result
↪': 68, 'message': '', 'dn': '', 'description': 'entryAlreadyExists'}]> received via
↪<ldap://openldap:389 - cleartext - user: cn=admin,o=test - bound - open - <local:␣
↪192.168.137.1:50397 - remote: 192.168.137.104:389> - tls not started - listening -␣
↪SyncStrategy>
DEBUG:ldap3:BASIC:done ADD operation, result <False>


# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50397 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - closed - <no socket> - tls not started - not listening -␣
↪SyncStrategy>
```

```
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:01.043802]
    Initial start time:  2015-06-05T23:38:29.505383
    Open socket time:    2015-06-05T23:38:29.505383
    Close socket time:   2015-06-05T23:38:30.549185
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 182
    Transmitted:         154
    Received:            28
  Messages:              5
    Transmitted:         3
    Received:            2
  Operations:            3
    Abandon:             0
    Bind:                1
    Add:                 1
    Compare:             0
    Delete:              0
    Extended:            0
    Modify:              0
    ModifyDn:            0
    Search:              0
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
  Restartable tries:     0
    Failed restarts:     0
    Successful restarts: 0
```

## The DELETE operation

The **Delete** operation allows a client to request the removal of an entry from the LDAP directory.

To perform a Delete operation you must specify the dn of the entry.

In the ldap3 library the signature for the Delete operation is:

```
def delete(self,
           dn,
           controls=None):
```

- dn: distinguished name of the object to delete
- controls: additional controls to send with the request

For synchronous strategies the delete method returns True if the operation was successful, returns False in case of errors. In this case you can inspect the result attribute of the connection object to get the error description.

For asynchronous strategies the add method returns the message id of the operation. You can get the operation result with the `get_response(message_id)` method of the connection object. If you use the `get_request=True` parameter you get the request dictionary back.

Only leaf entries (those with no subordinate entries) can be deleted with this operation.

You perform a Delete operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the Delete operation
c.delete('cn=user1,ou=users,o=company')
print(c.result)

# close the connection
c.unbind()
```

### Extended logging

To get an idea of what's happening when you perform a Delete operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=test', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
```

```
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'name': 'cn=admin,o=test', 'version': 3,
↪'authentication': {'sasl': None, 'simple': '<stripped 8 characters of sensitive␣
↪data>'}}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test -␣
↪unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.104:389> - tls␣
↪not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=test'
>>   authentication=AuthenticationChoice:
>>    simple=b'<stripped 8 characters of sensitive data>'
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'type': 'bindResponse', 'dn': '', 'saslCreds':␣
↪None, 'result': 0, 'referrals': None, 'message': '', 'description': 'success'}>␣
↪received via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:50024 - remote: 192.168.137.104:389> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>

# Performing the Delete operation:

DEBUG:ldap3:BASIC:start DELETE operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:DELETE request <{'entry': 'cn=user1,o=test'}> sent via <ldap://
↪openldap:389 - cleartext - user: cn=admin,o=test - bound - open - <local: 192.168.
↪137.1:50024 - remote: 192.168.137.104:389> - tls not started - listening -␣
↪SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  delRequest=b'cn=user1,o=test'
DEBUG:ldap3:NETWORK:sent 22 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
```

```
<<  delResponse=DelResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:DELETE response <[{'type': 'delResponse', 'dn': '', 'result': 0,
 ↪'referrals': None, 'message': '', 'description': 'success'}]> received via <ldap://
 ↪openldap:389 - cleartext - user: cn=admin,o=test - bound - open - <local: 192.168.
 ↪137.1:50024 - remote: 192.168.137.104:389> - tls not started - listening -␣
 ↪SyncStrategy>
DEBUG:ldap3:BASIC:done DELETE operation, result <True>


# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
 ↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
 ↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
 ↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
 ↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
 ↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.
 ↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
 ↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
 ↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
 ↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
 ↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
 ↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50024 - remote: 192.168.137.
 ↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
 ↪cn=admin,o=test - bound - closed - <no socket> - tls not started - not listening -␣
 ↪SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:01.387729]
    Initial start time:  2015-06-06T08:19:25.843569
    Open socket time:    2015-06-06T08:19:25.843569
    Close socket time:   2015-06-06T08:19:27.231298
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 94
    Transmitted:         66
    Received:            28
```

```
  Messages:               5
    Transmitted:          3
    Received:             2
  Operations:             3
    Abandon:              0
    Bind:                 1
    Add:                  0
    Compare:              0
    Delete:               1
    Extended:             0
    Modify:               0
    ModifyDn:             0
    Search:               0
    Unbind:               1
  Referrals:
    Received:             0
    Followed:             0
  Restartable tries:      0
    Failed restarts:      0
    Successful restarts: 0
```

## The MODIFY operation

The **Modify** operation allows a client to request the modification of an entry already present in the LDAP directory.

To perform a Modify operation you must specify the dn of the entry and the kind of changes requested.

In the ldap3 library the signature for the Modify operation is:

```
def modify(self,
           dn,
           changes,
           controls=None):
```

- dn: distinguished name of the object whose attributes must be modified
- changes: a dictionary of changes to be performed on the specified entry
- controls: additional controls to send in the request

For synchronous strategies the modify method returns True if the operation was successful, returns False in case of errors. In this case you can inspect the result attribute of the connection object to get the error description.

For asynchronous strategies the add method returns the message id of the operation. You can get the operation result with the `get_response(message_id)` method of the connection object. If you use the `get_request=True` parameter you get the request dictionary back.

There are 4 different kinds of change:

- MODIFY_ADD: add values listed to the specified attribute, creating the attribute if necessary.
- MODIFY_DELETE: delete values listed from the attribute. If no values are listed, or if all current values of the attribute are listed, the entire attribute is removed.
- MODIFY_REPLACE: replace all existing values of the specified attribute with the new values listed, creating the attribute if it did not already exist. A replace with no values will delete the entire attribute if it exists, and it is ignored if the attribute does not exist.

- MODIFY_INCREMENT: All existing values of the specified attribute are to be incremented by the listed value. The attribute must be appropriate for the request (e.g., it must have INTEGER or other increment-able values), and the modification must provide one and only one value. (RFC4525)

changes is a dictionary in the form {'attribute1': [(operation, [val1, val2, ...]), (operation2, [val1, val2, ...]), ...], 'attribute2': [(operation, [val1, val2, ...])], ...}

operation is MODIFY_ADD, MODIFY_DELETE, MODIFY_REPLACE, MODIFY_INCREMENT (import them from the ldap3 namespace)

The entire list of modifications is performed by the server in the order they are listed as a single atomic operation. While individual modifications may violate certain aspects of the directory schema (such as the object class definition and content rules), the resulting entry after the entire list of modifications is performed must conform to the requirements of the directory model and the controlling schema.

The Modify operation cannot be used to remove from an entry any of its distinguished values (the values which form the entry's relative distinguished name).

Due to the requirement for atomicity in applying the list of modifications in the Modify Request, the client may expect that no modifications have been performed if the Modify Response received indicates any sort of error, and that all requested modifications have been performed if the Modify Response indicates successful completion of the Modify operation.

You perform a Modify operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL, MODIFY_REPLACE

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')
c.bind()

# perform the Modify operation
c.modify('cn=user1,ou=users,o=company',
         {'givenName': [(MODIFY_REPLACE, ['givenname-1-replaced'])],
          'sn': [(MODIFY_REPLACE, ['sn-replaced'])]})
print(c.result)

# close the connection
c.unbind()
```

### Extended logging

To get an idea of what happens when you perform a Modify operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
```

```
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=test', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'version': 3, 'name': 'cn=admin,o=test',
↪'authentication': {'simple': '<stripped 8 characters of sensitive data>', 'sasl':␣
↪None}}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound␣
↪- open - <local: 192.168.137.1:52751 - remote: 192.168.137.104:389> - tls not␣
↪started - listening - SyncStrategy>
```

```
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=test'
>>   authentication=AuthenticationChoice:
>>    simple=b'<stripped 8 characters of sensitive data>'
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'message': '', 'description': 'success',
↪'referrals': None, 'saslCreds': None, 'result': 0, 'dn': '', 'type': 'bindResponse'}
↪> received via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:52751 - remote: 192.168.137.104:389> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>


# Performing the Modify operation:

DEBUG:ldap3:BASIC:start MODIFY operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:MODIFY request <{'entry': 'cn=user1,o=test', 'changes': [{
↪'attribute': {'type': 'givenName', 'value': ['givenname-1-replaced']}, 'operation':␣
↪2}, {'attribute': {'type': 'sn', 'value': ['sn-replaced']}, 'operation': 2}]}> sent␣
↪via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - bound - open -
↪<local: 192.168.137.1:52751 - remote: 192.168.137.104:389> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  modifyRequest=ModifyRequest:
>>   object=b'cn=user1,o=test'
>>   changes=Changes:
>>    Change:
>>     operation='replace'
>>     modification=PartialAttribute:
>>      type=b'givenName'
>>      vals=Vals:
>>       b'givenname-1-replaced'
>>    Change:
>>     operation='replace'
>>     modification=PartialAttribute:
>>      type=b'sn'
>>      vals=Vals:
>>       b'sn-replaced'
DEBUG:ldap3:NETWORK:sent 94 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  modifyResponse=ModifyResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:MODIFY response <[{'message': '', 'description': 'success',
↪'referrals': None, 'result': 0, 'dn': '', 'type': 'modifyResponse'}]> received via
↪<ldap://openldap:389 - cleartext - user: cn=admin,o=test - bound - open - <local:␣
↪192.168.137.1:52751 - remote: 192.168.137.104:389> - tls not started - listening -␣
↪SyncStrategy>
DEBUG:ldap3:BASIC:done MODIFY operation, result <True>


# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:52751 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - closed - <no socket> - tls not started - not listening -␣
↪SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:01.243813]
    Initial start time:  2015-06-10T18:23:50.618075
    Open socket time:    2015-06-10T18:23:50.618075
    Close socket time:   2015-06-10T18:23:51.861888
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 166
    Transmitted:         138
    Received:            28
  Messages:              5
    Transmitted:         3
    Received:            2
  Operations:            3
    Abandon:             0
    Bind:                1
    Add:                 0
    Compare:             0
    Delete:              0
    Extended:            0
    Modify:              1
    ModifyDn:            0
    Search:              0
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
  Restartable tries:     0
    Failed restarts:     0
    Successful restarts: 0
```

## The MODIFY-DN operation

The **ModifyDN** operation allows a client to change the Relative Distinguished Name (RDN) of an entry or to move an entry in the LDAP directory.

ModifyDN is really a two-flavours operation: you rename the last part of the dn **or** you move the entry in another container but you cannot perform both operations at the same time.

To perform a ModifyDN operation you must specify the dn of the entry and the new relative dn requested.

In the ldap3 library the signature for the Delete operation is:

```
def modify_dn(self,
              dn,
              relative_dn,
              delete_old_dn=True,
              new_superior=None,
              controls=None)
```

- dn: distinguished name of the entry whose relative name must be modified
- relative_dn: new relative dn of the entry
- delete_old_dn: remove the previous dn (defaults to True)
- new_superior: the new container of the entry
- controls: additional controls to send in the request

For synchronous strategies the modify_dn method returns True if the operation was successful, returns False in case of errors. In this case you can inspect the result attribute of the connection object to get the error description.

For asynchronous strategies the add method returns the message id of the operation. You can get the operation result with the `get_response(message_id)` method of the connection object. If you use the `get_request=True` parameter you get the request dictionary back.

You can rename an entry with a ModifyDN operation as in the following example (using the default synchronous strategy):

```
# import classes and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the ModifyDN operation to rename an entry
c.modify_dn('cn=user1,ou=users,o=company', 'cn=user2')
print(c.result)

# perform the ModifyDN operation to move an entry
c.modify_dn('cn=user2,ou=users,o=company', 'cn=user2', new_superior='ou=admins,
↪o=company')

print(c.result)

# close the connection
c.unbind()
```

### Extended logging

To get an idea of what's happening when you perform a ModifyDn operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP. AFter binding a rename is performed and then the entry is moved:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=test', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'authentication': {'sasl': None, 'simple':
↪'<stripped 8 characters of sensitive data>'}, 'version': 3, 'name': 'cn=admin,o=test
↪'}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:53484 - remote: 192.168.137.104:389> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=test'
>>   authentication=AuthenticationChoice:
>>    simple=b'<stripped 8 characters of sensitive data>'
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.168.
↪137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - unbound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'type': 'bindResponse', 'dn': '', 'referrals':␣
↪None, 'saslCreds': None, 'message': '', 'result': 0, 'description': 'success'}>␣
↪received via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:53484 - remote: 192.168.137.104:389> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>
```

```
# Starting the ModifyDN operation to perform a rename

DEBUG:ldap3:BASIC:start MODIFY DN operation via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:MODIFY DN request <{'newRdn': 'cn=user2', 'deleteOldRdn': True,
↪'entry': 'cn=user1,o=test', 'newSuperior': None}> sent via <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 -␣
↪remote: 192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  modDNRequest=ModifyDNRequest:
>>   entry=b'cn=user1,o=test'
>>   newrdn=b'cn=user2'
>>   deleteoldrdn='True'
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  modDNResponse=ModifyDNResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:MODIFY DN response <[{'type': 'modDNResponse', 'dn': '',
↪'referrals': None, 'message': '', 'result': 0, 'description': 'success'}]> received␣
↪via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - bound - open -
↪<local: 192.168.137.1:53484 - remote: 192.168.137.104:389> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:done MODIFY DN operation, result <True>


# Starting the ModifyDN operation to perform a move

DEBUG:ldap3:BASIC:start MODIFY DN operation via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:MODIFY DN request <{'newRdn': 'cn=user2', 'deleteOldRdn': True,
↪'entry': 'cn=user2,o=test', 'newSuperior': 'ou=moved,o=test'}> sent via <ldap://
↪openldap:389 - cleartext - user: cn=admin,o=test - bound - open - <local: 192.168.
↪137.1:53484 - remote: 192.168.137.104:389> - tls not started - listening -␣
↪SyncStrategy>
```

```
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  modDNRequest=ModifyDNRequest:
>>   entry=b'cn=user2,o=test'
>>   newrdn=b'cn=user2'
>>   deleteoldrdn='True'
>>   newSuperior=b'ou=moved,o=test'
DEBUG:ldap3:NETWORK:sent 54 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=3
<< protocolOp=ProtocolOp:
<<  modDNResponse=ModifyDNResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:MODIFY DN response <[{'type': 'modDNResponse', 'dn': '',
↪'referrals': None, 'message': '', 'result': 0, 'description': 'success'}]> received␣
↪via <ldap://openldap:389 - cleartext - user: cn=admin,o=test - bound - open -
↪<local: 192.168.137.1:53484 - remote: 192.168.137.104:389> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:done MODIFY DN operation, result <True>


# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <4> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
```

```
>> messageID=4
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:53484 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=test - bound - closed - <no socket> - tls not started - not listening -␣
↪SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:18.903383]
    Initial start time:  2015-06-11T22:03:36.180931
    Open socket time:    2015-06-11T22:03:36.180931
    Close socket time:   2015-06-11T22:03:55.084314
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 177
    Transmitted:         135
    Received:            42
  Messages:              7
    Transmitted:         4
    Received:            3
  Operations:            4
    Abandon:             0
    Bind:                1
    Add:                 0
    Compare:             0
    Delete:              0
    Extended:            0
    Modify:              0
    ModifyDn:            2
    Search:              0
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
  Restartable tries:     0
    Failed restarts:     0
```

## The SEARCH operation

The **Search** operation is used to request a server to return, subject to access controls and other restrictions, a set of entries matching a search filter. This can be used to read attributes from a single entry, from entries immediately subordinate to a particular entry, or from a whole subtree of entries.

In the ldap3 library the signature for the Search operation is:

```
def search(self,
           search_base,
           search_filter,
           search_scope=SUBTREE,
           dereference_aliases=DEREF_ALWAYS,
           attributes=None,
           size_limit=0,
           time_limit=0,
           types_only=False,
           get_operational_attributes=False,
           controls=None,
           paged_size=None,
           paged_criticality=False,
           paged_cookie=None):
```

- search_base: the base of the search request.
- search_filter: the filter of the search request. It must conform to the LDAP filter syntax specified in RFC4515.
- search_scope: specifies how broad the search context is:
  - BASE: retrieves attributes of the entry specified in the search_base.
  - LEVEL: retrieves attributes of the entries contained in the search_base. The base must reference a container object.
  - SUBTREE: retrieves attributes of the entries specified in the search_base and all subordinate containers downward.
- dereference_aliases: specifies how the server must treat references to other entries:
  - DEREF_NEVER: never dereferences entries, returns alias objects instead. The alias contains the reference to the real entry.
  - DEREF_SEARCH: while searching subordinates of the base object, dereferences any alias within the search scope. Dereferenced objects become the bases of further search scopes where the Search operation is also applied by the server. The server should eliminate duplicate entries that arise due to alias dereferencing while searching.
  - DEREF_BASE: dereferences aliases in locating the base object of the search, but not when searching subordinates of the base object.
  - DEREF_ALWAYS: always returns the referenced entries, not the alias object.
- attributes: a single attribute or a list of attributes to be returned by the search (defaults to None). If attributes is None no attribute is returned. If attributes is ALL_ATTRIBUTES or ALL_OPERATIONAL_ATTRIBUTES all user attributes or all operational attributes are returned.
- size_limit: maximum number of entries returned by the search (defaults to None). If None the whole set of found entries is returned, unless the server has a more restrictive rule.
- time_limit: number of seconds allowed for the search (defaults to None). If None the search can take an unlimited amount of time, unless the server has a more restrictive rule.
- types_only: doesn't return any attribute value, only the attribute names are returned.
- get_operational_attributes: if True returns informational attributes (managed automatically by the server) for each entry.
- controls: additional controls to send in the request.
- paged_size: if paged_size is greater than 0 a simple paged search is executed as described in RFC2696 (defaults to None). The search will return at most the specified number of entries.

- paged_criticality: if True the search will be executed only if the server is capable of performing a simple paged search. If False and the server is not capable of performing a simple paged search a standard search will be executed.
- paged_cookie: an *opaque* string received in a paged paged search that must be sent back while requesting subsequent entries of the search result.

### The LDAP filter

The LDAP filter defines the conditions that must be fulfilled in order for the Search to match a given entry and must follow the syntax defined in RFC 4515. The filter is composed of assertions that can be joined with AND (&) or OR (|) operators, or negated with the NOT (!) operator. The AND, OR, and NOT choices can be used to form combinations of assertions in a complex filter. At least one filter element must be present in an AND or in a OR.

An assertion is formed by 3 parts: the attribute name, a matching operator, and the matched value. The assertion itself is surrounded by parentheses, as in "(sn=Smith)". The LDAP schema defines how each attribute matching rule and ordering rule is evaluated.

An assertion is evaluated by the server to TRUE, FALSE, or Undefined. An assertion evaluates to Undefined when the server is not able to determine whether the assertion value matches an entry (for example when the matching rule is not defined for an attribute, when the type of filter is not implemented by the server or when the assertion value is invalid). If the filter evaluates to TRUE for a particular entry the attributes requested in the Search operation are returned for that entry as part of the Search result (subject to any applicable access control restrictions). If the filter evaluates to FALSE or Undefined, then the entry is ignored for the Search operation.

An AND is TRUE if all the assertion in the set evaluate to TRUE, FALSE if at least one assertion is FALSE and Undefined otherwise.

An OR is TRUE if at least one assertion evaluates to TRUE, FALSE if all the assertions in the set evaluate to FALSE and Undefined otherwise.

A NOT is TRUE if the assertion being negated is FALSE, FALSE if it is TRUE, and Undefined if it is Undefined.

If the search filter contains the following characters you must use the relevant escape ASCII sequence, as per RFC4515 (section 3): '*' -> '\\2A', '(' -> '\\28', ')' -> '\\29', '\' -> '\\5C', chr(0) -> '\\00'.

There are 7 match operators that can be used in a filter:

- EQUALITY (attribute=value): The matching rule for an equality filter is defined in the schema by the EQUALITY matching rule for the attribute type. The filter is TRUE when the EQUALITY rule returns TRUE as applied to the attribute and the asserted value.
- SUBSTRING (attribute=initial*middle*final): there can be at most one initial substring and at most one final substring of the assertion value, while there can be many middle substrings separated by an asterisk. The matching rule in a substrings filter is defined by the SUBSTR matching rule for the attribute type. The filter is TRUE when the SUBSTR rule returns TRUE as applied to the attribute and the asserted value.
- GREATER OR EQUAL (attribute>=value): The matching rule is defined by the ORDERING matching rule for the attribute type. The filter is TRUE when the ORDERING rule returns FALSE as applied to the attribute and the asserted value.
- LESS OR EQUAL (attribute<=value): The matching rules are defined by the ORDERING and EQUALITY matching rules for the attribute type. The filter is TRUE when either the ORDERING or EQUALITY rule returns TRUE as applied to the attribute and the asserted value.
- PRESENT (attribute=*): the filter is TRUE when there is an attribute present in an entry, FALSE when no attribute is present in an entry, and Undefined otherwise.
- APPROXIMATE (attribute~=value): the filter is TRUE when there is a value of the attribute type for which some server locally-defined approximate matching algorithm (e.g., spelling variations, phonetic match, etc.)

returns TRUE. If a value matches for equality, it also satisfies an approximate match. If approximate matching is not supported for the attribute, this filter should be treated as an equality filter by the server. The approximate algorithm, if available, is local to the server so you should check your server documentation to see if this matching operator can be used.

- EXTENSIBLE (attribute:=value): in the extensible filter the attribute part of the filter is augmented with other information (separated by a colon ":" as in "(o:dn:=Ace Industry)") to achieve particular search behaviours. Please check the documentation of your LDAP server to see what EXTENSIBLE syntax is available.

### NOT, AND and OR

You can negate the result of an assertion with the NOT (!) operator as in:

```
(!(sn=Smith))  # retrieves all entries where the sn is not Smith
```

You can join more than one assertion with the AND (&) and the OR (|) operator to create a complex filter. The AND and OR operator have their own parentheses that include all the assertion in the set:

```
(&(givenName=A*)(sn=Smith))  # retrieves all entries where sn attribute is Smith and␣
↪givenName starts with A.
(|(sn=Smith)(sn=Johnson))  # retrieves all entries where sn is Smith or Johnson.
```

You can even mix the NOT, AND and OR to form a more complex filter as in:

```
(|(&(objectClass=inetOrgPerson)(!(cn=Smith))(cn=admin*))  # retrieves all entries␣
↪whose cn starts with admin and all entries
of class inetOrgPerson with a surname different from Smith.
```

### Search scope and aliases

The scope of the search specifies how broad the search context will be. The LDAP database is a hierarchical structure (similar to a traditional file system) with a root and with container and leaf objects. a container can be stored in other containers, but not in a leaf object. It must be _clear that containers and leafs structure has nothing to do with the group and group membership objects. A group (groupOfNames) object is a leaf object with a member attribute that contains references to other objects.

The only way to know if an object is a container or a leaf object is to query the LDAP server schema for that object's class.

There are three possible scopes:

- BASE: the scope is constrained to the entry named by search_base.
- LEVEL: the scope is constrained to the immediate subordinates of the entry named by search_base.
- SUBTREE: the scope is constrained to the entry named by search_base and to all its subordinates.

An object can be an alias of another one defined somewhere in the LDAP data structure (it is similar to a link inside the LDAP database). While searching you can specify if "dereferencing" must be applied to the found aliases. The act of dereferencing an alias includes recursively dereferencing of aliases. The LDAP server should detect looping while dereferencing aliases in order to prevent denial-of-service attacks of this nature.

There are four possible ways of managing aliases while searching:

- DEREF_NEVER: never dereferences entries, returns alias objects instead. The alias contains the reference to the real entry.

- DEREF_SEARCH: while searching subordinates of the base object, dereferences any alias within the search scope. Dereferenced objects become the bases of further search scopes where the Search operation is also applied by the server. If the search scope is SUBTREE, the Search continues in the subtree of any dereferenced object. If the search scope is LEVEL, the search is applied to any dereferenced objects and is not applied to their subordinates. Servers should eliminate duplicate entries that arise due to alias dereferencing while searching.
- DEREF_BASE: dereferences aliases in locating the base object of the search, but not when searching subordinates of the base object.
- DEREF_ALWAYS: Dereference aliases both in searching and in locating the base object of the Search.

### Attributes

There are two kinds of attributes defined in the LDAP schema: User attributes and Operational Attributes. User attribute are added, modified and deleted with the usual LDAP operations, while Operational attributes are managed by the server and can only be read.

The server can apply some ACL to attributes and entries to prevent users from accessing data they don't have right to read or modify.

You can request a list of attributes to be returned for each found entry. You must specify the attribute names or the following values for attribute grouping:

- (ASTERISK): all user attributes, defined in ldap3.ALL_ATTRIBUTES
- (PLUS): all operational attributes, defined in ldap3.ALL_OPERATIONAL_ATTRIBUTES

1.1: no attributes, defined in ldap3.NO_ATTRIBUTES

Even if the RFC states that if you don't specify any attribute the server should return all available attributes for each entry found in the search the ldap3 library requires that you to specify at least one attribute in the attributes list, else it will perform a "1.1" Search request that returns only the dn of the entries found. To get all the user attributes you can use:

```
attributes=ldap3.ALL_ATTRIBUTES
```

while to get all the user and all the operational attributes you can use:

```
attributes=[ALL_ATTRIBUTES, ALL_OPERATIONAL_ATTRIBUTES]
```

Keep in mind that the server may not return some operational attribute if they are not explicitly requested (because they may take a long time or many resources to be computed), so if you need a specific attribute is better to request it explicitly. Also, some servers dont' return operational attributes information when reading the schema.

To request the operational attributes you can even set the get_operational_attributes parameter to True.

The standard LDAP RFCs define that if an attribute doesn't have a value it must not be returned at all. This can cause your code to become clumsy because you have to check always for existence of an attribute in the *attribute* dictionary of the response. To let the library return an empty attribute even if it is not present in the LDAP object retrieved by the search you can set the *return_empty_attributes* parameter to True in the Connection object, in this case all the requested attributes not present in the objects found by the search are set to an empty list.

### Checked Attributes

The checked attributes feature checks the LDAP syntax of the attributes defined in schema and returns a properly formatted entry result while performing searches. This means that if, for example, you have an attributes specified as GUID in the server schema you will get the properly formatted GUID value ('012381d3-3b1c-904f-b29a-012381d33b1c') in the connection.response[0]['attributes'] key dictionary instead of a sequence of bytes. Or if you

request an attribute defined as an Interger in the schema you will get the value already converted to int. Furthermore for attributes defined as single valued in schema you will get the value instead of a list of values (that would always be one sized). To activate this feature you must set the get info to GET_SCHEMA_INFO or GET_ALL_INFO value when defining the server object and the 'check_names' attributes to True in the Connection object (True by default).

There are a few of standard formatters defined in the library, most of them are defined in the relevants RFCs:

- format_unicode # returns an unicode object in Python 2 and a string in Python 3
- format_integer # returns an integer
- format_binary # returns a bytes() sequence
- format_uuid # returns a GUID (UUID) as specified in RFC 4122 - byte order is big endian
- format_uuid_le # same as above but byte order is little endian
- format_boolean # returns a boolean
- format_time # returns a datetime object (with properly defined timezone, or UTC if timezone is not specified) as defined in RFC 4517

You can even define your custom formatter for specific purposes. Just pass a dictionary in the format {'identifier': callable} in the 'formatter' parameter of the Server object. The callable must be able to receive a single byte value and convert it the relevant object or class instance.

The resolution order of the format feature is the following:

Custom formatters have precedence over standard formatter. In each category (from highest to lowest priority):

1. attribute name
2. attribute oid(from schema)
3. attribute names (from oid_info)
4. attribute syntax (from schema)

If a suitable formatter is not found the value will be rendered as bytes.

### Search constraints

You can limit the number of entries returned in a search or the time spent by the server performing the search using the size_limit and time_limit parameters. The server can also have some system wide constrains regarding the maximun number of entries returned in any search and the time spent in performing the search. It can also have some constrains on how the aliases are dereferenced. You must check the configuration of your LDAP server to verify which limitations are currenty active.

You can also request to not get any attribute value in the entries returned by the search with the types_only=True parameter.

---

**Note:** Microsoft Active Directory set an hard limit of 1000 entries returned by any search. So it's better to always set up a paged search when dealing with AD.

---

### Simple paged search

The search operation can perform a *simple paged search* as per RFC2696. You must specify the required number of entries returned in each response set. After the first search you must send back the cookie you get with each response in each subsequent search. If you send 0 as paged_size and a

valid cookie the search operation referred by that cookie is abandoned. Cookie can be found in connection.result['controls']['1.2.840.113556.1.4.319']['value']['cookie']; the server may return an estimated total number of entries in connection.result['controls']['1.2.840.113556.1.4.319']['value']['size']. You can change the paged_size in any subsequent search request.

Example:

```
from ldap3 import Server, Connection, SUBTREE
total_entries = 0
server = Server('test-server')
c = Connection(server, user='username', password='password')
c.search(search_base = 'o=test',
         search_filter = '(objectClass=inetOrgPerson)',
         search_scope = SUBTREE,
         attributes = ['cn', 'givenName'],
         paged_size = 5)
total_entries += len(c.response)
for entry in c.response:
    print(entry['dn'], entry['attributes'])
cookie = c.result['controls']['1.2.840.113556.1.4.319']['value']['cookie']
while cookie:
    c.search(search_base = 'o=test',
             search_filter = '(object_class=inetOrgPerson)',
             search_scope = SUBTREE,
             attributes = ['cn', 'givenName'],
             paged_size = 5,
             paged_cookie = cookie)
    total_entries += len(c.response)
    cookie = c.result['controls']['1.2.840.113556.1.4.319']['value']['cookie']
    for entry in c.response:
        print(entry['dn'], entry['attributes'])
print('Total entries retrieved:', total_entries)
```

Or you can use the much simpler extended operations package that wraps all this machinery and hides implementation details, you can choose to get back a generator or the whole list of entries found.

Working with a generator is better when you deal with very long list of entries or have memory issues:

```
# paged search wrapped in a generator
total_entries = 0
entry_generator = c.extend.standard.paged_search(search_base = 'o=test',
                                                 search_filter =
↪'(objectClass=inetOrgPerson)',
                                                 search_scope = SUBTREE,
                                                 attributes = ['cn', 'givenName'],
                                                 paged_size = 5,
                                                 generator=True)
for entry in entry_generator:
    total_entries += 1
    print(entry['dn'], entry['attributes'])
print('Total entries retrieved:', total_entries)
```

Remember that a generator can be consumed only one time, so you must elaborate the results in a sequential way.

Working with a list keeps all the found entries in a list and you can elaborate them in a random way:

```
# whole result list
entry_list = c.extend.standard.paged_search(search_base = 'o=test',
                                            search_filter =
↪'(objectClass=inetOrgPerson)',
```

```
                                            search_scope = SUBTREE,
                                            attributes = ['cn', 'givenName'],
                                            paged_size = 5,
                                            generator=False)
for entry in entry_list:
    print entry['attributes']
total_entries = len(entry_list)
print('Total entries retrieved:', total_entries)
```

If the paged search operation returns an error, the paged_search() method raises an Exception of class LDAPOperationResult, subclassed to the actual error returned by the server.

### Response

Responses are received and stored in the connection.response as a list of dictionaries. You can get the search result entries of a Search operation iterating over the response attribute. Each entry is a dictionary with the following field:

- dn: the distinguished name of the entry
- attributes: a dictionary of returned attributes and their values. Values are list. Values are in UTF-8 format
- raw_attributes: same as 'attributes' but not encoded (bytearray)

### Entries

Entries found in search are returned also in connection.entries as abstract.entry objects. This can be helpful when you use the ldap3 library from the interpreter prompt.

Each Entry object contains one object found in the search. You can access entry attributes either as a dictionary or as properties using the attribute name: entry['CommonName'] is the same of entry.CommonName and of entry.commonName or entry.commonname.

Each Entry has an entry_dn property that returns the distinguished name of the LDAP entry.

Attributes are stored in an internal dictionary with case insensitive access. You can even access the raw attribute with the entry_raw_attribute(attribute_name) to get an attribute raw value, or property entry_raw_attributes to get the whole raw attributes dictionary.

Entry is a read only object, you cannot modify or add any property to it. It's an iterable object that returns an attribute object at each iteration. Note that you get back the whole attribute object, not only the key as in a standard dictionary:

```
>>> c.entries[0]
DN: cn=person1,o=test
    cn: person1
    givenName: person1_givenname
    objectClass: inetOrgPerson
                 organizationalPerson
                 Person
                 ndsLoginProperties
                 Top
    sn: person1_surname
    GUID: fd9a0d90-15be-2841-fd82-fd9a0d9015be
```

and each attribute of the entry can be accessed as a dictionary or as a namespace:

```
>>> c.entries[0].GUID
    GUID: fd9a0d90-15be-2841-fd82-fd9a0d9015be
```

```
>>> c.entries[0].GUID.value
    'fd9a0d90-15be-2841-fd82-fd9a0d9015be'
>>> c.entries[0].GUID.raw_values
    [b'\xfd\x9a\r\x90\x15\xbe(A\xfd\x82\xfd\x9a\r\x90\x15\xbe']
>>> c.entries[0].GUID.values
    ['fd9a0d90-15be-2841-fd82-fd9a0d9015be']
```

An Entry can be converted to LDIF with the entry.entry_to_ldif() method and to JSON with the entry.entry_to_json() method. Entries can be easily printed at the interactive prompt:

```
>>> print(c.entries[0].entry_to_ldif())
version: 1
dn: cn=person1,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
ACL: 2#subtree#cn=person1,o=test#[All Attributes Rights]
ACL: 6#entry#cn=person1,o=test#loginScript
ACL: 2#entry#[Public]#messageServer
ACL: 2#entry#[Root]#groupMembership
ACL: 6#entry#cn=person1,o=test#printJobConfiguration
ACL: 2#entry#[Root]#networkAddress
sn: person1_surname
cn: person1
givenName: person1_givenname
GUID:: +J4sRRpsAEmjlfieLEUabA==
# total number of entries: 1

>>> print(c.entries[0].entry_to_json())
{
    "attributes": {
        "ACL": [
            "2#subtree#cn=person1,o=test#[All Attributes Rights]",
            "6#entry#cn=person1,o=test#loginScript",
            "2#entry#[Public]#messageServer",
            "2#entry#[Root]#groupMembership",
            "6#entry#cn=person1,o=test#printJobConfiguration",
            "2#entry#[Root]#networkAddress"
        ],
        "cn": [
            "person1"
        ],
        "givenName": [
            "person1_givenname"
        ],
        "GUID": [
            "f89e2c45-1a6c-0049-a395-f89e2c451a6c"
        ],
        "objectClass": [
            "inetOrgPerson",
            "organizationalPerson",
            "Person",
            "ndsLoginProperties",
            "Top"
        ],
        "sn": [
```

```
            "person1_surname"
        ]
    },
    "dn": "cn=person1,o=test"
}
```

To obtain already formatted values you must request the schema in the Server object with get_info=SCHEMA or get_info=ALL.

### Extended logging

To get an idea of what happens when you perform a Search operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:ERROR:hide sensitive data set to True
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=services', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'version': 3, 'authentication': {'sasl':␣
↪None, 'simple': '<stripped 8 characters of sensitive data>'}, 'name': 'cn=admin,
↪o=services'}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,o=services␣
↪- unbound - open - <local: 192.168.137.1:49445 - remote: 192.168.137.104:389> - tls␣
↪not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=services'
>>   authentication=AuthenticationChoice:
>>    simple=<stripped 8 characters of sensitive data>
DEBUG:ldap3:NETWORK:sent 41 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='invalidCredentials'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'message': '', 'description': 'invalidCredentials
↪', 'type': 'bindResponse', 'saslCreds': None, 'result': 49, 'dn': '', 'referrals':␣
↪None}> received via <ldap://openldap:389 - cleartext - user: cn=admin,o=services -␣
↪unbound - open - <local: 192.168.137.1:49445 - remote: 192.168.137.104:389> - tls␣
↪not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:done BIND operation, result <False>


# Performing the Search operation:

DEBUG:ldap3:BASIC:start SEARCH operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:SEARCH request <{'attributes': ['objectClass', 'sn'],
↪'dereferenceAlias': 3, 'filter': '(cn=test*)', 'timeLimit': 0, 'sizeLimit': 0,
↪'scope': 2, 'typesOnly': False, 'base': 'o=test'}> sent via <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=services - unbound - open - <local: 192.168.137.
↪1:49445 - remote: 192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  searchRequest=SearchRequest:
>>   baseObject=b'o=test'
>>   scope='wholeSubtree'
>>   derefAliases='derefAlways'
>>   sizeLimit=0
>>   timeLimit=0
>>   typesOnly='False'
>>   filter=Filter:
>>    substringFilter=SubstringFilter:
>>     type=b'cn'
>>     substrings=Substrings:
>>      Substring:
>>       initial=b'test'
>>   attributes=AttributeSelection:
>>    b'objectClass'    b'sn'
DEBUG:ldap3:NETWORK:sent 63 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 114 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  searchResEntry=SearchResultEntry:
<<   object=b'cn=testSASL,o=test'
<<   attributes=PartialAttributeList:
<<    PartialAttribute:
<<     type=b'sn'
```

```
<<     vals=Vals:
<<      b'testSASL'
<<    PartialAttribute:
<<     type=b'objectClass'
<<     vals=Vals:
<<      b'inetOrgPerson'      b'organizationalPerson'      b'person'      b'top'
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  searchResDone=SearchResultDone:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:SEARCH response entry <{'attributes': {'sn': ['testSASL'],
↪'objectClass': ['inetOrgPerson', 'organizationalPerson', 'person', 'top']}, 'dn':
↪'cn=testSASL,o=test', 'type': 'searchResEntry', 'raw_attributes': {'sn': [b'testSASL
↪'], 'objectClass': [b'inetOrgPerson', b'organizationalPerson', b'person', b'top']}}>
↪ received via <ldap://openldap:389 - cleartext - user: cn=admin,o=services -␣
↪unbound - open - <local: 192.168.137.1:49445 - remote: 192.168.137.104:389> - tls␣
↪not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done SEARCH operation, result <True>


# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:49445 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:01.126074]
    Initial start time:  2015-07-16T07:40:46.871386
    Open socket time:    2015-07-16T07:40:46.871386
    Close socket time:   2015-07-16T07:40:47.997460
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 253
    Transmitted:         111
    Received:            142
  Messages:              6
    Transmitted:         3
    Received:            3
  Operations:            3
    Abandon:             0
    Bind:                1
    Add:                 0
    Compare:             0
    Delete:              0
    Extended:            0
    Modify:              0
    ModifyDn:            0
    Search:              1
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
  Restartable tries:     0
    Failed restarts:     0
    Successful restarts: 0
```

## The COMPARE operation

The **Compare** operation allows a client to request the comparison of an entry attribute against a specific value.

To perform a Compare operation you must specify the dn of the entry, the name of the attribute and the value to compare.

In the ldap3 library the signature for the Compare operation is:

```
    def compare(self,
            dn,
            attribute,
            value,
            controls=None):

* dn: distinguished name of the entry whose attribute is to compare

* attribute: name of the attribute to compare
```

```
* value: value to be compared

* controls: additional controls to send in the request
```

For synchronous strategies the compare method returns True if the attribute value equals the value sent in the operation, returns False in case it's different.

For asynchronous strategies the add method returns the message id of the operation. You can get the operation result with the `get_response(message_id)` method of the connection object. If you use the `get_request=True` parameter you get the request dictionary back.

False value is returned even if the entry is not found in the LDAP server. You can check the description of the connection result attribute to know the reason of the missed match. The dessription is set to compareTrue, compareFalse, or an appropriate error.

compareTrue indicates that the specified value matches a value of the attribute according to the attribute's EQUALITY matching rule. compareFalse indicates that the specified value and the value (or values) of the attribute does not match.

Note that some LDAP servers may establish access controls that permit the values of certain attributes 0to be compared but not read by other means (as for the userPassword attribute).

You perform a Compare operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the Compare operation
comparison = c.compare('cn=user1,ou=users,o=company', 'sn', 'surname')
print(comparison)

# close the connection
c.unbind()
```

### Extended logging

To get an idea of what happens when you perform a Compare operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:ERROR:hide sensitive data set to True
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
```

```
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=services', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'name': 'cn=admin,o=services',
↪'authentication': {'sasl': None, 'simple': '<stripped 8 characters of sensitive␣
↪data>'}, 'version': 3}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,
↪o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=services'
>>   authentication=AuthenticationChoice:
>>    simple=<stripped 8 characters of sensitive data>
DEBUG:ldap3:NETWORK:sent 41 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='invalidCredentials'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'description': 'invalidCredentials', 'message': '
↪', 'type': 'bindResponse', 'saslCreds': None, 'result': 49, 'dn': '', 'referrals':␣
↪None}> received via <ldap://openldap:389 - cleartext - user: cn=admin,o=services -␣
↪unbound - open - <local: 192.168.137.1:51287 - remote: 192.168.137.104:389> - tls␣
↪not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <False>


# Performing the Compare operation:

DEBUG:ldap3:BASIC:start COMPARE operation via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:COMPARE request <{'entry': 'cn=user1,o=test', 'attribute': 'sn',
↪'value': 'surname'}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,
↪o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  compareRequest=CompareRequest:
```

```
>>   entry=b'cn=user1,o=test'
>>   ava=AttributeValueAssertion:
>>    attributeDesc=b'sn'
>>    assertionValue=b'surname'
DEBUG:ldap3:NETWORK:sent 39 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 20 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  compareResponse=CompareResponse:
<<   resultCode='noSuchObject'
<<   matchedDN=b'o=test'
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:COMPARE response <[{'description': 'noSuchObject', 'message': '',
↪ 'type': 'compareResponse', 'result': 32, 'dn': 'o=test', 'referrals': None}]>␣
↪received via <ldap://openldap:389 - cleartext - user: cn=admin,o=services - unbound␣
↪- open - <local: 192.168.137.1:51287 - remote: 192.168.137.104:389> - tls not␣
↪started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done COMPARE operation, result <False>


# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
Connection Usage:
  Time: [elapsed:         0:00:01.040277]
    Initial start time:  2015-07-16T07:38:39.883408
    Open socket time:    2015-07-16T07:38:39.883408
    Close socket time:   2015-07-16T07:38:40.923685
  Server:
    Servers from pool:   0
    Sockets open:        1
    Sockets closed:      1
    Sockets wrapped:     0
  Bytes:                 121
    Transmitted:         87
    Received:            34
  Messages:              5
    Transmitted:         3
    Received:            2
  Operations:            3
    Abandon:             0
    Bind:                1
    Add:                 0
    Compare:             1
    Delete:              0
    Extended:            0
    Modify:              0
    ModifyDn:            0
    Search:              0
    Unbind:              1
  Referrals:
    Received:            0
    Followed:            0
  Restartable tries:     0
    Failed restarts:     0
    Successful restarts: 0
```

## The ABANDON operation

The use of the **Abandon** operation is very limited. Its intended function is to allow a client to request a server to give up an uncompleted operation. Since there is no response from the server the client cannot tell the difference between a successfully abandoned operation and a completed operation. The Bind, Unbind and the Abandon operations cannot be abandoned.

Clients should not send multiple Abandon requests for the same message ID and must be prepared to receive a response to the original operation, in case the server has already completed the request before receiving the Abandon request.

The only (and indirect) way to know if the operation has been abandoned is to wait for a reasonable time to see if you get the response to the original operation (the one that has to be abandoned).

The Abandon operation can be used with asynchronous strategies only because you need the message ID that is returned by the operation method of the connection object.

The Abandon Request is defined as follows:

```
def abandon(self,
            message_id,
            controls=None):


* message_id: id of the previously sent request to be abandoned

* controls: additional controls to send in the request
```

The abandon method returns True if the abandon request was sent, returns False if the request cannot be sent (for Bind, Unbind and Abandon requests).

You perform an Abandon operation as in the following example (using the asynchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL, ASYNC

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password', client_strategy=ASYNC)

# perform a Delete operation
message_id = c.delete('cn=user1,ou=users,o=company')

# perform the Abandon operation
c.abandon(message_id)

# check if the operation to be abandoned has been already executed
result = connection.get_response(message_id)
if not result:
    print('Abandon successful')
else:
    print('Too late... Cannot abandon')

# close the connection
c.unbind()
```

An alternative use of the Abandon operation is to send an Abandon(0) message, that can be used as a keep-alive mechanism, even if this is not indicated in the RFCs.

## The EXTENDED operation

The **Extended** operation allows a client to request an operation that may not be defined in the current RFCs but is available on the server.

To perform a Compare operation you must specify the *name* of the extended operation (defined with an OID) and the value to be sent.

In the ldap3 library the signature for the Extended operation is:

```
def extended(self,
             request_name,
             request_value=None,
             controls=None,
```

```
            no_encode=None)

* request_name: name of the extended operation

* request_value: optional value sent in the request (defaults to None)

* controls: additional controls to send in the request

* no_encode: when True the value is passed without any encoding (defaults to False)
```

Some common extended operation are defined in the Connection.extend namespace.

You perform an Extended operation as in the following example (using the default synchronous strategy):

```
# import class and constants
from ldap3 import Server, Connection, ALL

# define the server
s = Server('servername', get_info=ALL)  # define an unsecure LDAP server, requesting␣
↪info on DSE and schema

# define the connection
c = Connection(s, user='user_dn', password='user_password')

# perform the Extended operation
result = c.extended('0.1.2.3', 'my_value')
print(result)

# close the connection
c.unbind()
```

### Extended logging

To get an idea of what happens when you perform an Extended operation this is the extended log from a session to an OpenLdap server from a Windows client with dual stack IP:

```
# Initialization:

INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:ERROR:hide sensitive data set to True
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=389, use_
↪ssl=False, get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=389, use_ssl=False, get_info='NO_INFO'), user='cn=admin,o=services', password='
↪<stripped 8 characters of sensitive data>', auto_bind='NONE', version=3,␣
↪authentication='SIMPLE', client_strategy='SYNC', auto_referrals=True, check_
↪names=True, collect_usage=True, read_only=False, lazy=False, raise_
↪exceptions=False)>
DEBUG:ldap3:NETWORK:opening connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldap://openldap:389 - cleartext> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldap://openldap:389 - cleartext>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪389)]> with mode IP_V6_PREFERRED


# Opening the connection (trying IPv6 then IPv4):

DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 389, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldap://openldap:389 -␣
↪cleartext - user: cn=admin,o=test - unbound - closed - <local: [::]:50396 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 389)]
DEBUG:ldap3:NETWORK:connection open for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>


# Authenticating to the LDAP server with the Simple Bind method:

DEBUG:ldap3:BASIC:start BIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'name': 'cn=admin,o=services',
↪'authentication': {'sasl': None, 'simple': '<stripped 8 characters of sensitive␣
↪data>'}, 'version': 3}> sent via <ldap://openldap:389 - cleartext - user: cn=admin,
↪o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.168.137.
↪104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
```

```
>>   version=3
>>   name=b'cn=admin,o=services'
>>   authentication=AuthenticationChoice:
>>    simple=<stripped 8 characters of sensitive data>
DEBUG:ldap3:NETWORK:sent 41 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='invalidCredentials'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'description': 'invalidCredentials', 'message': '
↪', 'type': 'bindResponse', 'saslCreds': None, 'result': 49, 'dn': '', 'referrals':␣
↪None}> received via <ldap://openldap:389 - cleartext - user: cn=admin,o=services -␣
↪unbound - open - <local: 192.168.137.1:51287 - remote: 192.168.137.104:389> - tls␣
↪not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <False>


# Performing the Extended operation:

TODO

# Closing the connnection (via the Unbind operation):

DEBUG:ldap3:BASIC:start UNBIND operation via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldap://openldap:389 - cleartext -␣
↪user: cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote:␣
↪192.168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - open - <local: 192.168.137.1:51287 - remote: 192.
↪168.137.104:389> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:NETWORK:connection closed for <ldap://openldap:389 - cleartext - user:␣
↪cn=admin,o=services - unbound - closed - <no socket> - tls not started - not␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

These are the usage metrics of this session:

```
TODO
```

# Extend namespace

LDAP standard operations are quite simple, so the protocol allows the LDAP server vendor to add Extended operations used to perform more complex operation. You can use the Connection.extended() operation to execute any operation defined by the LDAP server vendor but it's hard to properly define the "payload" of the operation because you must describe the operation parameters in the ASN.1 notation as requested by the server and encode them with BER encoding.

The *extend* namespace of the Connection object includes a predefined set of extended (or complex) LDAP operations that can be performed in a simple way. The namespace is partitioned in sub-namespaces, the first is for extended operations defined in the standard LDAP RFCs, the others groups operations for specific kinds of LDAP server.

You can call the requested operation and get the extended result back as specified in the relevant RFC or documentation The result dictionary is augmented with the specific keys returned by the extended response.

To use an extended operation just call it in the usual way, the payload is properly encoded and decoded. for example:

```
c = Connection(...)
c.bind()
i_am = c.extend.standard.who_am_i()
```

You get the response value as the return value of the function and as an additional field of the `result` dictionary.

## Extended standard RFCs operations

The extend.standard namespace contains extended operation defined in current RFCs:

```
extend.standard
    extend.standard.who_am_i()
    extend.standard.modify_password(
        user,
        old_password,
        new_password,
        hash_algorithm=None,
        salt=None
    )
    extend.standard.paged_search(search_base,
        search_filter,
        search_scope,
        dereference_aliases,
        attributes,
        size_limit,
        time_limit,
        types_only,
```

```
        get_operational_attributes,
        controls,
        paged_size,
        paged_criticality,
        generator
    )
    extend.standard.persistent_search(
        connection,
        search_base,
        search_filter,
        search_scope,
        dereference_aliases,
        attributes,
        size_limit,
        time_limit,
        controls,
        changes_only,
        events_type,
        notifications,
        streaming,
        callback
    )
```

To get the identity of the bound user:

```
c = Connection(....)
c.bind()
i_am = c.extend.standard.who_am_i()
```

if who_am_i() returns an empty string an anonymous connection is bound.

To modify a user password:

```
from ldap3 import Server, Connection, HASHED_SALTED_SHA256
s = Server(...)
c = Connection(s, ...)
c.bind()  # bind as someone that has permission to change user's password
new_password = c.extend.standard.modify_password('cn=test1,o=test', 'old_password',
↪'new_password', HASHED_SALTED_SHA256)  # a new password is set, hashed with sha256␣
↪and a random salt
```

A special case with modify_password is for LDAP servers that follow RFC3062. If you send the old password and do not specify a new password, the server should generate a new password compliant with the server password policy:

```
s = Server(...)
c = Connection(s, ...)
c.bind()  # bind as someone that has permission to change user's password
new_password = c.extend.standard.modify_password('cn=test1,o=test', 'old_password')
↪# a new password is generated by the server if compliant with RFC3062
```

The extend.standard.paged_search() operation is a convenient wrapper for the simple paged search as specified in the RFC2696. You can indicate how many entries will be read in the paged_size parameter (defaults to 100) and you get back a *generator* for the entries. If you set to False the generator parameter of the search will be fully executed before returning the results. If generator is set to True (the default) any subsequent search will be executed only when you read all the previous read entries, saving memory.

In the modify_password() extended operation you can specify an hashing algorithm, if your LDAP server use hashed password but don't compute the hash by itself. Otherwise you can send the password and the server will hash it.

Algorithms names are defined in the ldap3 module. You can choose between:

- HASHED_NONE (no hashing is performed, password is sent in plain text)
- HASHED_MD5
- HASHED_SHA
- HASHED_SHA256
- HASHED_SHA384
- HASHED_SHA512
- HASHED_SALTED_MD5
- HASHED_SALTED_SHA
- HASHED_SALTED_SHA256
- HASHED_SALTED_SHA384
- HASHED_SALTED_SHA512

If you don't specify a *salt* parameter a random salt will be generated by the ldap3 library. Keep in mind that only salted password can provide a strong level of security against dictionary attacks.

To directly modify the userPassword attribute via a modify operation you probably have to send the hashed password. In this case you can use the hashed() function in the ldap3.utils.hashed package:

```
from ldap3 import HASHED_SALTED_SHA
from ldap3.utils.hashed import hashed

hashed_password = hashed(HASHED_SALTED_SHA, 'new_password')
c.modify('cn=user1,o=test', {'userPassword': [(MODIFY_REPLACE,[hashed_password])]})
```

To enable Persistent Searches to get all modification in the tree as they happens (for logging purpose):

```
from ldap3 import Server, Connection, ASYNC_STREAM
s = Server('myserver')
c = Connection(s, 'cn=admin,o=resources', 'password', client_strategy=ASYNC_STREAM)
c.stream = open('myfile.log', 'w+)
p = c.extend.standard.persistent_search()
```

now the persistent search is running in an internal thread. Each modification is recorded in the log in LDIF-CHANGE format, with the event type, event time and the modified dn and changelog number (if available) as comments.

This uses the AsyncStream Strategy, because the Persistent Search never sends the "SearchDone" packet, and this is not a valid LDAP3 behaviour. This is the reason for which the https://www.ietf.org/proceedings/50/I-D/ldapext-psearch-03.txt draft has never been approved as a standard RFC. The AsyncStream strategy sends each received packet to an external thread where it can be processed as soon as it is received.

In the persistent_search() method you can use the same parameter of a standard search. It also accepts some additional parameters specific of the persistent search:

```
def persistent_search(self,
                      search_base='',
                      search_filter='(objectclass=*)',
                      search_scope=SUBTREE,
                      dereference_aliases=DEREF_NEVER,
                      attributes=ALL_ATTRIBUTES,
                      size_limit=0,
                      time_limit=0,
```

```
                        controls=None,
                        changes_only=True,
                        show_additions=True,
                        show_deletions=True,
                        show_modifications=True,
                        show_dn_modifications=True,
                        notifications=True,
                        streaming=True,
                        callback=None
                        ):
```

If you don't pass any parameters the search should be globally applied in your LDAP server.

You can choose which kind of events to show with the `show_*` boolean parameters. `notification=True` allows you to receive the original dn of a modify_dn operation and the changelog number if provided by the server.

If you want to stop the persistent search you can use `p.stop()`. Use `p.start()` to start it again.

If you don't provide a stream (a file to write to), a StringIO object is used. You can use it as a standard file or get the value of the StringIO object with `c.stream.getvalue()`.

For example an output from my test suite is the following:

```
# 2016-07-10T23:34:41.616615
# add
dn: cn=[71973491]modify-dn-1,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
sn: modify-dn-1
cn: [71973491]modify-dn-1
ACL: 2#subtree#cn=[71973491]modify-dn-1,o=test#[All Attributes Rights]
ACL: 6#entry#cn=[71973491]modify-dn-1,o=test#loginScript
ACL: 2#entry#[Public]#messageServer
ACL: 2#entry#[Root]#groupMembership
ACL: 6#entry#cn=[71973491]modify-dn-1,o=test#printJobConfiguration
ACL: 2#entry#[Root]#networkAddress

# 2016-07-10T23:34:41.888506
# modify dn
# previous dn: cn=[71973491]modify-dn-1,o=test
dn: cn=[71973491]modified-dn-1,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
sn: modify-dn-1
cn: [71973491]modified-dn-1
ACL: 2#subtree#cn=[71973491]modified-dn-1,o=test#[All Attributes Rights]
ACL: 6#entry#cn=[71973491]modified-dn-1,o=test#loginScript
ACL: 2#entry#[Public]#messageServer
ACL: 2#entry#[Root]#groupMembership
ACL: 6#entry#cn=[71973491]modified-dn-1,o=test#printJobConfiguration
ACL: 2#entry#[Root]#networkAddress

# 2016-07-10T23:34:41.929022
```

```
# delete
dn: cn=[71973491]modified-dn-1,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
sn: modify-dn-1
cn: [71973491]modified-dn-1
ACL: 2#subtree#cn=[71973491]modified-dn-1,o=test#[All Attributes Rights]
ACL: 6#entry#cn=[71973491]modified-dn-1,o=test#loginScript
ACL: 2#entry#[Public]#messageServer
ACL: 2#entry#[Root]#groupMembership
ACL: 6#entry#cn=[71973491]modified-dn-1,o=test#printJobConfiguration
ACL: 2#entry#[Root]#networkAddress
```

If you call the persistent_search() method with straming=False you can get the modified entries with the p.next() method. Each call to p.next() returns one event, with the extended control already decoded (as dict values) if available.

If you call the persistent_search() method with **callback=myfynction** (where myfunction is a callable, including lambda, accepting a dict as parameter) your function will be called for each event received in the persistent serach. The function will be called in the same thread of the persistent search, so it should not block.

## Extended Novell operations

Novell extended operations are specific for the Novell (NetIq) eDirectory LDAP Server and return and set the universal password of the specified user:

```
extend.novell
    extend.novell.get_bind_dn()
    extend.novell.get_universal_password(user)
    extend.novell.set_universal_password(user, new_password)
    extend.novell.start_transaction()
    extend.novell.end_transaction(commit=True, controls)
    extend.novell.add_members_to_groups(members, groups, fix=True, transaction=True)
    extend.novell.remove_members_from_groups(members, groups, fix=True,␣
↪transaction=True)
    extend.novell.check_groups_memberships(members, groups, fix=False,␣
↪transaction=True)
    extend.novell.list_replicas()
    extend.novell.partition_entry_count()
    extend.novell.replica_info()
```

## Extended Microsoft operations

Microsoft extended operations are intended for Active Directory:

```
extend.microsoft
    extend.microsoft.dir_sync(sync_base, sync_filter, attributes, cookie, object_
↪security, ancestors_first, public_data_only, incremental_values, max_length, hex_
↪guid)
    extend.microsoft.modify_password(user, new_password, old_password=None)
```

# Encoding

The LDAP RFCs states that strings sent and received by the LDAP server must be encoded with the **utf-8** encoding. Data in ldap3 flow from different sources to different tergets, so they can have different encodings. There are 6 different flows in the ldap3 library:

- data coming from the server
- data going to the server
- data coming from the user
- data going to the user
- data going to output logs
- data coming from flaky servers that return the DN or the schema in not utf-8 encoding.

## Server data flow

The LDAP protocol stores strings in a **Directory String** type that should always be in **utf-8** and it's stored in the `DEFAULT_CLIENT_ENCODING` config parameter. For flaky server this encoding can be changed with:

```
>>> from ldap3 import set_config_parameter
>>> set_config_parameter('DEFAULT_SERVER_ENCODING', 'latin-1')
```

DEFAULT_SERVER_ENCODING can be changed multiple times as needed. To know the current `DEFAULT_SERVER_ENCODING` you can use the following code:

```
>>> from ldap3 import get_config_parameter
>>> get_config_parameter('DEFAULT_SERVER_ENCODING')
'utf-8'
```

Some servers don't completely follow the LDAP RFCs and send data in a different encoding or in a mix of encodings. For example Active Directory can send the DN of entries found in a search in a different encoding than utf-8. In this case you can use the `ADDITIONAL_SERVER_ENCODINGS` config parameter to decode the DN of the Search operation response. It can be set to one encoding or a list of encodings. If a list of encodings is provided ldap3 tries sequentially each encoding until a valid decode is performed. If any of the specified encodings is able to decode the value then an `UnicodeError` exception is raised.

`ADDITIONAL_SERVER_ENCODINGS` defaults to ['latin1, 'koi8-r'] for european and russian encodings

## User data flow

ldap3 can receive data from the user in 4 different ways:

- data input during a program execution
- data stored in the source files
- data typed at the Python interpreter (the usual >>>)
- data input from another program via stdin

When dealing with user input the encoding can (and probably is) different from utf-8, so ldap3 tries to guess what encoding is used and store it in the `DEFAULT_CLIENT_ENCODING` parameter. ldap3 uses the **stdin.encoding** if present, else the sys.getdefaultencoding() and if neither is present sets `DEFAULT_CLIENT_ENCODING` to 'utf-8'.

You can set a specific `DEFAULT_CLIENT_ENCODING` with the following code:

```
>>> from ldap3 import set_config_parameter
>>> set_config_parameter('DEFAULT_CLIENT_ENCODING', 'my_encoding')
```

DEFAULT_CLIENT_ENCODING can be changed multiple times as needed. To know the current `DEFAULT_CLIENT_ENCODING` you can use the following code:

```
>>> from ldap3 import get_config_parameter
>>> get_config_parameter('DEFAULT_CLIENT_ENCODING')
'utf-8'
```

This parameter is used when the `auto_encode` and `auto_escape` parameters of the Connection object are set to `True`.

ldap3 can send data from the user in 3 different ways:

- data printed at the stdout
- data printed at the console
- data printed in the log files

Data printed at stdout and at the console (for example representation of ldap3 Server and Connection objects) are encoded with **stdout.encoding** if present, else the `ascii` encoding is used. If ldap3 is unable to properly represents the data with the selected encoding then it uses a hexadecimal representation of the byte value of the data. This ensure that there are no error and no missing data at the REPL or console level. This can lead to some differences in output between Python 3 and Python 2.

### Log file data flow

Data printed in the log files are always encoded in `ascii` with a `backslashreplace` failback in case of unprintable ascii values. This should assure that the log is written without any decoding error.

### Raw data in search response

You can always access the raw byte values of attributes returned in a search accessing the `raw_attributes` key of a response entry or the `raw_values` property of an Entry object in the Abstraction Layer.

## Abstraction Layer

**A more pythonic LDAP**: LDAP operations look clumsy and hard-to-use because they reflect the old-age idea that time-consuming operations should be done on the client to not clutter and hog the server with unneeded elaboration. ldap3 includes a fully functional **Abstraction Layer** that lets you interact with the DIT in a modern and *pythonic* way. With the Abstraction Layer you don't need to directly issue any LDAP operation at all.

### Overview

With the Abstraction Layer you describe LDAP objects using the ObjectDef and AttrDef classes and access the LDAP server via a *Cursor* in read-only or read-write mode. Optionally you can use a Simplified Query Language to read the Entries from the DIT.

All classes can be imported from the ldap3 package:

```
from ldap3 import ObjectDef, AttrDef, Reader, Writer, Entry, Attribute,␣
↪OperationalAttribute
```

The Abstraction Layer relies on a simple **ORM** (Object Relational Mapping) that links Entries object to entries stored in the LDAP. Each Entry object refers to an ObjectDef that describes the relation between the Attributes stored in the Entry and the attributes stored in the DIT.

## ObjectDef class

The ObjectDef class is used to define an abstract Entry object. You can create ObjectDefs manually, defining each Attribute defininition (AttrDef) or in an automatic way with the information read from the schema.

To automatically create an ObjectDef just use the following code on an open connection where the schema has been read by the server:

```
>>> person = ObjectDef(['inetOrgPerson'], connection)
>>> person
OBJ: inetOrgPerson [inetOrgPerson OID: 2.16.840.1.113730.3.2.2, organizationalPerson␣
↪OID: 2.5.6.7, person OID: 2.5.6.6, top OID: 2.5.6.0]
MUST: cn, objectClass, sn
MAY: audio, businessCategory, carLicense, departmentNumber, description,␣
↪destinationIndicator, displayName, employeeNumber, employeeType,
     facsimileTelephoneNumber, givenName, homePhone, homePostalAddress, initials,␣
↪internationalISDNNumber, jpegPhoto, l, labeledURI, mail,
     manager, mobile, o, ou, pager, photo, physicalDeliveryOfficeName, postOfficeBox,␣
↪postalAddress, postalCode, preferredDeliveryMethod,
     preferredLanguage, registeredAddress, roomNumber, secretary, seeAlso, st, street,
↪ telephoneNumber, teletexTerminalIdentifier, telexNumber,
     title, uid, userCertificate, userPKCS12, userPassword, userSMIMECertificate,␣
↪x121Address, x500UniqueIdentifier
```

As you can see the *person* object has been populated with all attributes from the hierarchy of classes starting from *inetOrgPerson* up to *top*. Mandatory attributes (MUST) are listed separately from optional (MAY) attributes.

For each attribute you get additional information useful to interact with it:

```
>>> person.sn
ATTR: sn - mandatory: True - single_value: False
  Attribute type: 2.5.4.4
    Short name: sn, surName
    Single value: False
    Superior: name
    Equality rule: caseIgnoreMatch
    Syntax: 1.3.6.1.4.1.1466.115.121.1.15 [('1.3.6.1.4.1.1466.115.121.1.15', 'LDAP_
↪SYNTAX', 'Directory String', 'RFC4517')]
    Mandatory in: person
    Optional in: RFC822localPart, mozillaAbPersonAlpha
    Extensions:
      X-ORIGIN: RFC 4519
      X-DEPRECATED: surName
    OidInfo: ('2.5.4.4', 'ATTRIBUTE_TYPE', ['sn', 'surname'], 'RFC4519')
```

When manually creating a new ObjectDef instance you can specify the LDAP class(es) of the entries you will get back in a search. The object class(es) will be automatically added to the query filter:

```
person = ObjectDef('inetOrgPerson')
engineer = ObjectDef(['inetOrgPerson', 'auxEngineer'])
```

Once you have defined an ObjectDef instance you can add the attributes definition with the `add()` method of ObjectDef. You can also use the += operator as a shortcut. AttrDef(s) can be removed with the `remove()` method or using the −= operator.

ObjectDef is an iterable that returns each AttrDef object (the whole AttrDef object, not only the key). AttrDefs can be accessed either as a dictionary or as a property, spaces are removed and keys are not case sensitive:

```
cn_attr_def = person['Common Name']
cn_attr_def = person['commonName']  # same as above
cn_attr_def = person.CommonName  # same as above
```

This eases the use at the interactive >>> prompt where you don't have to remember the case of the attribute name. *Autocompletion feature* is enabled, so you can get a list of all defined attributes as property just pressing TAB at the interactive prompt.

Each class has a useful representation that summarize the instance status. You can access it directly at the interactive prompt, or in a program with the str() function.

## AttrDef class

The AttrDef class is used to define an abstract LDAP attribute. If you use the automatic ObjectDef creation the relevant AttrDefs are automatically created. AttrDef has a single mandatory parameter, the attribute name, and a number of optional parameters. The optional `key` parameter defines a friendly name to use while accessing the attribute. The `description` parameter can be used for storing additional information on the Attribute. When defining only the attribute name you can add it directly to the ObjectDef (the AttrDef is automatically defined):

```
cn_attribute = AttrDef('cn, description='This is the internal account name')
person.add(cn_attribute)

person += AttrDef('cn', description='This is the internal account name')  # same as␣
↪above
person += 'cn'  # same as above, without description
```

You can even add a list of attrDefs or attribute names to an ObjectDef:

```
person += [AttrDef('cn', key = 'Common Name'), AttrDef('sn', key = 'Surname')]
person += ['cn', 'sn']  # as above, but keys are the attribute names
```

### Validation

You can specify a `validate` parameter to check if the attribute value is valid. Two parameters are passed to the callable, the AttrDef.key and the value. The callable must return a boolean allowing or denying the validation:

```
deps = {'A': 'Accounting', 'F': 'Finance', 'E': 'Engineering'}
# checks that the parameter in query is in a specific range
valid_department = lambda attr, value: True if value in deps.values() else False
person += AttrDef('employeeType', key = 'Department', validate = validDepartment)
```

In this example the Cursor object will raise an exception if values for the 'Department' are not 'Accounting', 'Finance' or 'Engineering'.

### Pre Query transformation

A `pre_query` parameter indicates a callable used to perform a transformation on the value to be searched for the attribute defined:

```
# transform value to be search
def get_department_code(attr, value):
    for dep in deps.items():
        if dep[1] == value:
            return dep[0]
    return 'not a department'

person += AttrDef('employeeType', key = 'Department', pre_query = get_department_code)
```

When you perform a search with 'Accounting', 'Finance' or 'Engineering' for the Department key, the real search will be for employeeType = 'A', 'F' or 'E'.

### Post query transformation

A 'post_query' parameter indicates a callable to perform a transformation on the returned value:

```
get_department_name = lambda attr, value: deps.get(value, 'not a department') if attr␣
↪== 'Department' else value
person += AttrDef('employeeType', key = 'Department', post_query = get_department_
↪name)
```

When you have an 'A', an 'F', or an 'E' in the employeeType attribute you get 'Accounting', 'Finance' or 'Engineering' in the 'Department' property of the Person entry.

With a multivalue attribute post_query receives a list of all values in the attribute. You can return an equivalent list or a single string.

### Dereferencing DNs

With `dereference_dn` you can establish a relation between different ObjectDefs. When dereference_dn is set to an ObjectDef the Cursor reads the attribute and use its value as a DN for an object to be searched (using a temporary Reader) with the specified ObjectDef in the same Connection. The result of the second search is returned as value of the first search:

```
department = ObjectDef('groupOfNames')
department += 'cn'
department += AttrDef('member', key = 'employeer', dereference_dn = person)  # values␣
↪of 'employeer' will be the 'Person' entries members of the found department
```

## Cursor

There are two kind of *Cursor* in the Abstraction Layer, **Reader** and **Writer**. This helps to avoid the risk of accidentally change values when you're just reading them. This is a safe-guard because many application uses LDAP only for reading information, so having a read-only Cursor eliminates the risk of accidentally change or remove an entry. A Writer Cursor cannot read data from the DIT as well, Writer cursors are only used for DIT modification. Please refer to the Abstraction Layer tutorial for an in-depth description of Cursor capabilities and usage.

### Reader Cursor

Once you have defined the ObjectDef(s) and the AttrDef(s) you can instance a Reader for the ObjectDef. With it you can perform searches using a standard LDAP filter or a simplified query language (explained in next paragraph). To execute a different search the reader can be reset to its initial status with the `reset()` method.

A Reader cursor has the following attributes:

- connection: the connection to use.
- definition: the ObjectDef used by the Reader instance.
- query: the simplified query. It can be a standard LDAP filter (see next paragraph).
- base: the DIT base where to start the search.
- components_in_and: defines if the query components are in AND (True, default) or in OR (False).
- sub_tree: specifies if the search must be performed through the whole subtree (True, default) or only in the specified base (False).
- get_operational_attributes: specifies if the search must return the operational attributes (True) of found entries. Defaults to False.
- controls: optional controls to use in the search operation.
- attributes: the list of the attributes requested
- execution_time: the last time the query has run
- schema: the server schema, if any
- entries: the Entries returned by the Search operation
- operations: a list of LDAP Operation performed in the last Cursor operation
- errors: a list of LDAP Operation unsuccessful in the last Cursor operation
- failed: a boolean that indicates if any LDAP operation failed in the last Cursor operation

To perform a search Operation you can use any of the following methods:

- search(): standard search.
- search_level(): force a Level search.
- search_subtree(): force a whole sub-tree search, starting from 'base'.
- search_object(): force a object search, DN to search must be specified in 'base'.
- search_paged(page_size, criticality): perform a paged search, with 'page_size' number of entries for each call to this method. If 'criticality' is True the server aborts the operation if the Simple Paged Search extension is not available, else return the whole result set.

To retrieve some matching entries from a search operation the cursor:

- match_dn(dn): returns a list of entries where the specified text is found in the dn. The match is case insensitive
- match(attributes, value): returns a list of entries where the specified text is found in one of the attribute values. The match is case insensitive and checks for single and multi-valued attributes. The `attributes` parameter can be an attribute name or a list of attribute names

Example:

```
s = Server('server')
c = Connection(s, user = 'username', password = 'password')
query = 'Department: Accounting'  # explained in next paragraph
person_reader = Reader(c, person, 'o=test', query)
person_reader.search()
```

The result of the search will be found in the `entries` property of the `person_reader` object.

A Reader object is an iterable that returns the entries found in the last search performed. It also has a useful representation that summarize the Reader configuration and status:

```
print(personReader)
CONN   : ldap://server:389 - cleartext - user: cn=admin,o=test - version 3 - unbound -
↪ closed - not listening - SyncWaitStrategy
BASE   : 'o=test' [SUB]
DEFS   : 'inetOrgPerson' [CommonName <cn>, Department <employeeType>, Surname <sn>]
QUERY  : 'Common Name :test-add*, surname:=t*' [AND]
PARSED : 'CommonName: =test-add*, Surname: =t*' [AND]
ATTRS  : ['cn', 'employeeType', 'sn', '+'] [OPERATIONAL]
FILTER : '(&(objectClass=inetOrgPerson)(cn=test-add*)(sn=t*))'
ENTRIES: 1 [SUB] [executed at: Sun Feb  9 20:43:47 2014]
```

### Writer Cursor

A Writer Cursor has no Search capability because it can be only used to create new Entries or to modify the Entries in a Reader cursor or in an LDAP Search operation.

Instead of the search_* methods the Writer has the following methods:

- from_cursor: creates a Writer cursor from a Reader cursor, populated with a copy of the Entries in the Reader cursor
- from_response: create a Writer cursor from a Search operation response, populated with a copy of the Entries in the Search response
- commit: writes all the pending changes to the DIT
- discard: discards all the pending changes
- new: creates a new Entry
- refresh_entry: re-reads the Entry from the DIT

## Simplified Query Language

In the Reader you can express the query filter using the standard LDAP filter syntax or using a *Simplified Query Language* that resembles a dictionary structure. If you use the standard LDAP filter syntax you must use the real attribute names because the filter is directly passed to the Search operation.

The Simplified Query Language filter is a string of key-values couples separated with a ‘,’ (comma), in each of the couples the left part is the attribute key defined in an AttrDef object while the right part is the value (or values) to be searched. Parts are separed with a ‘:’ (colon). Keys can be prefixed with a ‘&’ (AND) or a ‘|’ (OR) for searching all the values or at least one of them. Values can be prefixed with an optional exclamation mark ‘!’ (NOT) for negating the search followed by the needed search operator (‘=’, ‘<’, ‘>’, ‘~’). The default operator is ‘=’ and can be omitted. Multiple values are separated by a ‘;’ (semi-colon).

A few examples:

```
'CommonName: bob' -> (cn=bob)
'CommonName: bob; john; michael' -> (|(cn=bob)(cn=john)(cn=michael))
'Age: > 21' -> (age>=21)
'&Age: > 21; < 65' ->&(age<=65)(age>=21))
'Department: != Accounting'' -> (!(EmployeeType=A))
'|Department:Accounting; Finance' -> (|(EmployeeType=A)(EmployeeType=C))
```

There are no parentheses in the Simplified Query Language, this means that you cannot mix components with ‘&’ (AND) and ‘|’ (OR). You have the ‘component_in_and’ flag in the Reader object to specify if components are in ‘&’ (AND, True value) or in ‘|’ (OR, False value). ‘component_in_and’ defaults to True:

```
'CommonName: b*, Department: Engineering' -> (&(cn=b*)(EmployeeType=E'))
```

Object classes defined in the ObjectDef are always included in the filter, so for the previous example the resulting filter is:

```
(&(&(objectClass=inetOrgPerson)(objectClass=AuxEngineer))(cn=b*)(EmployeeType=E))
```

when using a Reader with the 'engineer' ObjectDef.

## Entry

Cursors contains Entries that are the Python representation of entries stored in the LDAP DIT. There are two types of Entries, **Read** and **Writable**. Each Entry has a `state` attribute that keeps information on the current status of the Entry.

Entries are returned as the result of a Search operation or a Reader search. You can access entry attributes either as a dictionary or as properties using the AttrDef key you specified in the ObjectDef. `entry['CommonName']` is the same of `entry.Common Name` of `entry.CommonName` of `entry.commonName` and of `entry.commonname`.

Each Entry has a entry_dn() method that returns the distinguished name of the LDAP entry, and a entry_cursor() method that returns a reference to the Cursor used to read the entry.

Attributes are stored in an internal dictionary with case insensitive access by the key defined in the AttrDef. You can access the raw attribute with the `entry_raw_attribute(attribute_name)` to get an attribute raw value, or `entry_raw_attributes()` to get the whole raw attributes dictionary.

Because Attribute names are used as Entry class attributes all the "operational" attributes and method of an entry starts with **entry**_. An Entry as the following attributes and methods:

- entry_dn: the DN of the LDAP entry
- entry_cursor: the cursor object the Entry belongs to
- entry_status: a description of the current status of the Entry (can be any of 'Initial', 'Virtual', 'Missing mandatory attributes', 'Read', 'Writable', 'Pending changes', 'Committed', 'Ready for deletion', 'Ready for moving', 'Ready for renaming', 'Deleted').
- entry_definition: the ObjectDef (with relevant AttrDefs) of the Entry
- entry_raw_attributes: raw attribute values as read from the DIT
- entry_mandatory_attributes: the list of attributes that are mandatory for this Entry
- entry_attributes: formatted attribute values read from the DIT
- entry_attributes_as_dict: a dictonary with formatted attribute value
- entry_read_time: the time of last read of the Entry from the LDAP server
- entry_raw_attribute(attribute): method to request a specific raw attribute
- entry_to_json(raw=False, indent=4, sort=True, stream=None, checked_attributes=True): method to convert an Entry to a JSON representation
- entry_to_ldif(all_base64=False, line_separator=None, sort_order=None, stream=None): method to convert an Entry to a LDIF representation

A Read Entry has the following additional method:

- entry_writable(object_def=None, writer_cursor=None, attributes=None, custom_validator=None): method to create a new Writable Entry *linked* to the original Entry. This means that every change to the Entry is reflected to the original one

A Writable Entry has the following additional properties and methods:

- entry_virtual_attributes: list of the available attributes without a value
- entry_commit_changes(refresh=True, controls=None): writes all pending changes to the DIT
- entry_discard_changes(): discards all pending changes
- entry_delete(): set the entry for deletion (performed at commit time)
- entry_refresh(self, tries=4, seconds=2): re-reads the Entry attribute values from the LDAP Server
- entry_move(destination_dn): set the entry for moving (performed at commit time)
- entry_rename(new_name): set the entry for renaming (performed at commit time)

An Entry can be converted to LDIF with the `entry_to_ldif()` method and to JSON with the `entry_to_json()` method. Entries can be easily printed at the interactive prompt:

```
>>> print(c.entries[0].entry_to_ldif())
version: 1
dn: cn=person1,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
sn: person1_surname
cn: person1
givenName: person1_givenname
GUID:: +J4sRRpsAEmjlfieLEUabA==
# total number of entries: 1

>>> print(c.entries[0].entry_to_json())
{
    "attributes": {
        "cn": [
            "person1"
        ],
        "givenName": [
            "person1_givenname"
        ],
        "GUID": [
            "f89e2c45-1a6c-0049-a395-f89e2c451a6c"
        ],
        "objectClass": [
            "inetOrgPerson",
            "organizationalPerson",
            "Person",
            "ndsLoginProperties",
            "Top"
        ],
        "sn": [
            "person1_surname"
        ]
    },
    "dn": "cn=person1,o=test"
}
```

## Attribute

Values found for each attribute are stored in the Attribute object. You can access the 'values' and the 'raw_values' lists. You can also get a reference to the relevant AttrDef in the 'definition' property, and to the relevant Entry in the 'entry' property. You can iterate over the Attribute to get each value:

```
person_common_name = person_entry.CommonName
for cn in person_common_name:
    print(cn)
    print(cn.raw_values)
```

If the Attribute has a single value you get it in the 'value' property. This is useful while using the Python interpreter at the >>> interactive prompt. If the Attribute has more than one value you get the same 'values' list in 'value'. When you want to assign the attribute value to a variable you must use 'value' (or 'values' if you always want a list):

```
my_department = person_entry.Department.value
```

When an entry is Writable the Attribute has additional attributes and methods and operators used to apply changes to the attribute values:

- virtual: True if the attribute is new and still not stored in the DIT
- changes: the list of the pending changes for the attribute
- add(value): adds one or more values to the attribute, same of **+=**
- set(value): sets one or more values for the attribute, removing any previous stored value, same of **=**
- delete(value): delete one or more values from the attribute, same of **-=**
- remove(): sets the attribute for deletion
- discard(): discards all pending changes in the Attribute

## Modifying an Entry

With the Abstraction Layer you can "build" your Entry object and then commit it to the LDAP server in a simple pythonic way. First you must obtain a **Writable** Entry. Entry may become writable in four different way: as Entries from a Reader Cursor, as Entries form a Search response, as a single Entry from a Search response or as a new (Virtual) Entry:

```
>>> # this example is at the >>> prompt. Create a connection and a Reader cursor for␣
↪the inetOrgPerson object class
>>> from ldap3 import Connection, Reader, Writer, ObjectDef
>>> c = Connection('sl10', 'cn=my_user,o=my_org', 'my_password', auto_bind=True)
>>> o = ObjectDef('inetOrgPerson', c)  # automatic read of the inetOrgPerson␣
↪structure from schema
>>> r = Reader(c, o, 'o=test')  # we don't need to provide a filter because of the␣
↪objectDef implies '(objectclass=inetOrgPerson)'
>>> r.search()  # populate the reader with the Entries found in the Search

# make a Writable Cursor from the person_reader Reader Cursor
>>> w = Writer.from_cursor(r)
>>> e = w[0]  # A Cursor is indexed on the Entries collection

# make a Writable Cursor from an LDAP search response, you must specify the objectDef
```

```
>>> c.search('o=test', '(objectClass=inetOrgPerson), attributes=['cn', 'sn',
↪'givenName']
>>> w = Writer.from_response(c, c.response, 'inetOrgPerson')
>>> e = w[0]

# make a Writable Entry from the first entry of an LDAP search response, an implicit␣
↪Writer Cursor is created
>>> e = c.entries[0].entry_writable()

# make a new Writable Entry. The Entry remains in "Virtual" state until committed to␣
↪the DIT
>>> e = w.new('cn=new_entry, o=test')
```

Now you can use the `e` Entry object as a Python class object with standard behaviour:

```
>>> e.sn += 'Young'  # add an additional value to an existing attribute
>>> e.givenname = 'John'  # create a new attribute and assign a value to it -␣
↪attribute is flagged 'Virtual' until commit
>>> e
DN: cn=smith_j,o=test - STATUS: Writable, Pending changes - READ TIME: 2016-10-
↪19T09:51:08.919905
    cn: smith_j
    givenName: <Virtual>
                CHANGES: [('MODIFY_REPLACE', ['John'])]
    objectClass: inetOrgPerson
                 organizationalPerson
                 Person
                 ndsLoginProperties
                 Top
    sn: Smith
        CHANGES: [('MODIFY_ADD', ['Young'])]
```

Now let's perform the commit of the Entry and check the refreshed data:

```
>>> e.entry_commit_changes()
True
>>> e
DN: cn=smith_j,o=test - STATUS: Writable, Committed - READ TIME: 2016-10-19T09:54:58.
↪321715
    cn: [05038763]modify-dn-2
    givenName: John
    objectClass: inetOrgPerson
                 organizationalPerson
                 Person
                 ndsLoginProperties
                 Top
    sn: Smith
        Young
```

As you can see the status of the entry is "Writable, Committed" and the read time has been updated.

For specific types (boolean, integers and dates) you can set the value to the relevant Python type. The ldap3 library will perform the necessary conversion to the value expected from the LDAP server.

You can discard the pending changes with `e.entry_discard_changes()` or delete the whole entry with `e.delete()`. You can also move the Entry to another container in the DIT with `e.entry_move()` or renaming it with `e.entry_rename)`.

### Matching entries in cursor results

Once a cursor is populated with entries you can get a specific entry with the standard index feature of List object: `r.entries[0]` returns the first entry found, `r.entries[1]` returns he second one and any subsequent entry is returned by the relevant index number. The Cursor object has a shortcut for this operation: you can use `r[0]`, `r[1]` (and so on) to perform the same operation. Furthermore, the Cursor object has an useful feature that helps you to find a specific entry without knowing its index: when you use a string as the Cursor index the text will be searched in all entry DNs. If only one entry matches it is returned, if more than one entry match the text a KeyError exception is raised. You can also use the `r.match_dn(dn)` method to return all entries with the specified text in the DN and `r.match(attributes, value)` to return all entries that contain the `value` in any of the specified `attributes` where you can pass a single attribute name or a list of attribute names. When searching for values the either the formatted attribute and the raw value are checked.

### OperationalAttribute

The OperationalAttribute class is used to store Operational Attributes read with the 'get_operational_attributes' of the Reader object set to True. It's the same of the Attribute class except for the 'definition' property that is not present. Operational attributes key are prefixed with '**OA_**'.

## LDIF (LDAP Data Interchange Format)

LDIF is a data interchange format for LDAP. It is defined in RFC2849 (June 2000) in two different flavours: *LDIF-CONTENT* and *LDIF-CHANGE*. LDIF-CONTENT is used to describe LDAP entries in an ASCII stream (i.e. a file), while LDIF-CHANGE is used to describe Add, Delete, Modify and ModifyDn operations. *These two formats have different purposes and cannot be mixed in the same stream.* If the DN of the entry or an attribute value contains any unicode character the value must be base64 encoded, as specified in RFC2849. ldap3 is compliant to the latest LDIF format (version: 1).

ldap3 can stream the output of a search (LDIF-CONTENT) or of an operation (LDIF-CHANGE) to a file object. It cannot read a file with LDIF content because this is not defined in the LDAPv3 standard. Each LDAP server has its own tools to import an LDIF-CONTENT or to perform LDIF-CHANGE.

### LDIF-CONTENT

You can use the LDIF-CONTENT flavour with any search result:

```
...
# request a few objects from the ldap server
result = c.search('o=test','(cn=test-ldif*)', SUBTREE, attributes = ['sn',
↪'objectClass'])
ldif_output = c.response_to_ldif()
...
```

ldif_output will contain:

```
version: 1
dn: cn=test-ldif-1,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
```

```
sn: test-ldif-1

dn: cn=test-ldif-2,o=test
objectClass: inetOrgPerson
objectClass: organizationalPerson
objectClass: Person
objectClass: ndsLoginProperties
objectClass: Top
sn:: dGVzdC1sZGlmLTItw6DDssO5


# total number of entries: 2
```

you can even request a LDIF-CONTENT for a response you saved early:

```
# request a few objects from the ldap server
response1 = c.search('o=test','(cn=test-ldif*)', SUBTREE, attributes = ['sn',
↪'objectClass'])
response2 = c.search('o=test','(!(cn=test-ldif*))', SUBTREE, attributes = ['sn',
↪'objectClass'])
ldif_output = c.response_to_ldif(response1)
```

ldif_output will contain the LDIF representation of the response entries.

## LDIF-CHANGE

To get the LDIF representation of Add, Modify, Delete and ModifyDn operation you must use the LDIF strategy. With this strategy operations are not executed on an LDAP server but are converted to an LDIF-CHANGE format that can be sent to an LDAP server.

For example:

```
from ldap3 import Connection, LDIF
connection = Connection(server = None, client_strategy = LDIF)  # no need of real␣
↪LDAP server
connection.add('cn=test-add-operation,o=test'), 'inetOrgPerson',
               {'objectClass': 'inetOrgPerson', 'sn': 'test-add', 'cn': 'test-add-
↪operation'})
```

in connection.response you will find:

```
version: 1
dn: cn=test-add-operation,o=test
changetype: add
objectClass: inetorgperson
sn: test-add
cn: test-add-operation
```

A more complex modify operation (from the RFC2849 examples):

```
from ldap3 import MODIFY_ADD. MODIFY_DELETE, MODIFY_REPLACE
connection.modify('cn=Paula Jensen,ou=Product Development,dc=airius,dc=com',
    {'postaladdress': (MODIFY_ADD, ['123 Anystreet $ Sunnyvale, CA $ 94086']),
     'description': (MODIFY_DELETE, []),
     'telephonenumber': (MODIFY_REPLACE, ['+1 408 555 1234', '+1 408 555 5678']),
     'facsimiletelephonenumber': (MODIFY_DELETE, ['+1 408 555 9876'])
    })
```

```
returns:

version: 1
dn: cn=Paula Jensen,ou=Product Development,dc=airius,dc=com
changetype: modify
add: postaladdress
postaladdress: 123 Anystreet $ Sunnyvale, CA $ 94086
-
delete: description
-
replace: telephonenumber
telephonenumber: +1 408 555 1234
telephonenumber: +1 408 555 5678
-
delete: facsimiletelephonenumber
facsimiletelephonenumber: +1 408 555 9876
-
```

## Streaming the output to a file

When producing LDIF-CONTENT output you can have all operation results in a single stream. To get this simply set the stream attribute of the Connection to a stream object (for example to a file) and *open* the connection. If you don't specify the stream object a StringIO will be used. You can get the value with the c.stream.getvalue() method:

```
from ldap3 import Connection, LDIF
c = Connection(None, client_strategy=LDIF)
with c:
    c.delete('cn=test1, o=test')
    c.delete('cn=test2, o=test')
    result = c.stream.getvalue()  # needed because the stream is closed when the
↪connection exits the context
```

result will be:

```
version: 1

dn: cn=test1,o=test
changetype: delete

dn: cn=test2,o=test
changetype: delete
```

If you just define a file object as stream you'll find the output in the file:

```
c = Connection(None, client_strategy=LDIF)
c.stream = open('output.ldif', 'w')
with c:
    c.delete('cn=test1, o= test')
    c.delete('cn=test2, o=test')
```

you will find the LDIF output in the output.ldif file.

When producing LDIF-CONTENT you can pass an existing stream object to the response_to_ldif() method to add the LDIF output to the stream. If the stream is empty the ldif version header will be added.

### Custom line separator

The LDIF stream uses the default line separator (os.linesep) of the system where ldap3 is running as line separator in the LDIF stream. If you need a different line separator you can specify it in the *c.strategy.line_separator* attribute:

```
c.strategy.line_separator = '\\r\\n'
```

### Customizable descriptor order

RFC 2849 doesn't specify any specific order for the lines in the LDIF output except than *version: 1* in the first line of the stream. The library starts any new record with the dn and all subsequent *descriptor: value* lines are in the order they are received by the library. This should no be an issue with an LDIF import in another system, but if you have problems you can force a specific order for the descriptors in any of the LDIF operation: To achieve this you must set the c.strategy.order attribute to a dict where the keys are set to the names of the operations you want their resulting descriptor order is changed and the value to a list of descriptor. The LDIF output lines will be ordered following the order of the descriptor in the list. For example if you add to the previous code:

```
c.strategy.order = dict(delRequest = ['changetype:', 'dn:'])
```

you will get:

```
version: 1

changetype: delete
dn: cn=test1,o=test

changetype: delete
dn: cn=test2,o=test
```

The possible operation names are: addRequest, delRequest, modifyRequest, modDNRequest.

To change the order of a searchRequest just pass the list in the requested order.

## Exceptions

The ldap3 exceptions hierarchy includes a LDAPException root class with two main branches: LDAPExceptionError and LDAPExceptionResult. The LDAPExceptionError contains 42 different exceptions that should help understanding what's going wrong when you have an error. A few of these (including all the LDAPCommunicationError exceptions) have multiple inheritance either from the LDAPExceptionError and from specific Python exceptions. This let you choose between catching standard Python errors or the more detailed ldap3 errors. The LDAPCommunicationError exceptions will be created at runtime and inherit the specific socket exception raised by the interpreter, so you can catch them with the specific socket.error or the more general LDAPCommunicationError.

LDAPOperationResult (the other branch of LDAPException hierarchy) includes 48 exceptions, one for each possible error result (except RESULT_SUCCESS) specified in the LDAPv3 protocol. When you create a connection object with the "raise_exceptions" parameter set to True any unsuccesful LDAP operation will throw an exception of this class, subclassed to the specific LDAP result code exception. For example if you get an INVALID_DN_SYNTAX (result code 34) the connection will raise the LDAPInvalidDNSyntaxResult exception, with the following parameter: result, description, dn, message and the response_type. You can specify which result codes you don't want to raise exceptions, the default is : RESULT_COMPARE_FALSE, RESULT_COMPARE_TRUE, RESULT_REFERRAL (and of course RESULT_SUCCESS). You can change this behaviour in the ldap3 __init__.py package or at runtime modifying the ldap3.DO_NOT_RAISE_EXCEPTIONS list.

The "raise_exceptions" mode is helpful if you want exceptions to flow up in the code and manage them at a upper level than the single operation level. This mode works in every kind of strategy, even in the ReusableStrategy (for connection pooling) where exceptions are trapped in the "effective" connection thread and are sent back to the calling connection object in the main (or another) thread.

LDAPException

–LDAPExceptionError

—-LDAPBindError

—-LDAPCertificateError

—-LDAPChangeError (also inherits from ValueError)

—-LDAPCommunicationError (all may inherit from socket.error)

——LDAPReferralError

——LDAPSessionTerminatedByServer

——LDAPSocketCloseError

——LDAPSocketOpenError

——LDAPSocketReceiveError

——LDAPSocketSendError

——LDAPUnknownRequestError

——LDAPUnknownResponseError

—-LDAPConfigurationError

——LDAPUnknownStrategyError

——LDAPUnknownAuthenticationMethodError

——LDAPSSLConfigurationError

——LDAPDefinitionError

—-LDAPConnectionIsReadOnlyError

—-LDAPConnectionPoolNameIsMandatoryError

—-LDAPConnectionPoolNotStartedError

—-LDAPControlError (also inherits from ValueError)

—-LDAPExtensionError

—-LDAPInvalidDereferenceAliasesError (also inherits from ValueError)

—-LDAPInvalidFilterError

—-LDAPInvalidPort

—-LDAPInvalidScopeError (also inherits from ValueError)

—-LDAPInvalidServerError

—-LDAPKeyError (also inherits from KeyError)

—-LDAPLDIFError

—-LDAPMaximumRetriesError

—-LDAPMetricsError

—-LDAPObjectClassError (also inherits from ValueError)

—-LDAPObjectError

—-LDAPPasswordIsMandatoryError

—-LDAPCursorError

—-LDAPMaximumRetriesError

—-LDAPSASLBindInProgressError

—-LDAPSASLMechanismNotSupportedError

—-LDAPSASLPrepError

—-LDAPSchemaError

—-LDAPServerPoolError

—-LDAPServerPoolExhaustedError

—-LDAPSSLNotSupportedError (also inherits from ImportError)

—-LDAPStartTLSError

—-LDAPTypeError

–LDAPOperationResult

—-LDAPAdminLimitExceededResult

—-LDAPAffectMultipleDSASResult

—-LDAPAliasDereferencingProblemResult

—-LDAPAliasProblemResult

—-LDAPAssertionFailedResult

—-LDAPAttributeOrValueExistsResult

—-LDAPAuthMethodNotSupportedResult

—-LDAPAuthorizationDeniedResult

—-LDAPBusyResult

—-LDAPCanceledResult

—-LDAPCannotCancelResult

—-LDAPConfidentialityRequiredResult

—-LDAPConstraintViolationResult

—-LDAPEntryAlreadyExistsResult

—-LDAPESyncRefreshRequiredResult

—-LDAPInappropriateAuthenticationResult

—-LDAPInappropriateMatchingResult

—-LDAPInsufficientAccessRightsResult

—-LDAPInvalidAttributeSyntaxResult

—-LDAPInvalidCredentialsResult

—-LDAPInvalidDNSyntaxResult

—-LDAPLCUPInvalidDataResult

—-LDAPLCUPReloadRequiredResult

—-LDAPLCUPResourcesExhaustedResult

—-LDAPLCUPSecurityViolationResult

—-LDAPLCUPUnsupportedSchemeResult

—-LDAPLoopDetectedResult

—-LDAPNamingViolationResult

—-LDAPNoSuchAttributeResult

—-LDAPNoSuchObjectResult

—-LDAPNoSuchOperationResult

—-LDAPNotAllowedOnNotLeafResult

—-LDAPNotAllowedOnRDNResult

—-LDAPObjectClassModsProhibitedResult

—-LDAPObjectClassViolationResult

—-LDAPOperationsErrorResult

—-LDAPOtherResult

—-LDAPProtocolErrorResult

—-LDAPReferralResult

—-LDAPSASLBindInProgressResult

—-LDAPSizeLimitExceededResult

—-LDAPStrongerAuthRequiredResult

—-LDAPTimeLimitExceededResult

—-LDAPTooLateResult

—-LDAPUnavailableCriticalExtensionResult

—-LDAPUnavailableResult

—-LDAPUndefinedAttributeTypeResult

—-LDAPUnwillingToPerformResult

# LDAP3 Utils

## DN parsing

tbd

## Password hashing

tbd

### URI parsing

tbd

### Case Insensitive Dictionary

tbd

### Conversion helpers

In the ldap3.utils.conv you can find some helpful function to apply encoding to/from unicode:

- `to_unicode(obj, encoding=None)`: converts a byte object to unicode, if no encoding is specified with the `encoding` parameter the function tries to use the local environment encoding.
- `to_raw((obj, encoding='utf-8')`: converts a string to bytes, using the specified encoding (default is utf-8)
- `escape_filter_chars(text, encoding=None)`: escapes a string for using as a value in a filter assertion. Escaping is defined in RFC4515. If no encoding is defined tries to use the local environment encoding.
- *escape_bytes(bytes_value)*': applies LDAP escaping to a sequence of byte values so that it can be used in an LDAP Add or Modify operation.

These functions can be used in Python 2 or Python 3 with no changes in the code.

## Logging

ldap3 has an extended logging capability that uses the standard Python logging libray and integrates with the logging facility of the client application.

To enable logging the application must have a working logging configuration that emits logging at the DEBUG level:

```
import logging
logging.basicConfig(filename='client_application.log', level=logging.DEBUG)
```

This is intended to avoid the mix of ldap3 logging records in the application logging. Only when you set the log level of your application to DEBUG ldap3 starts to emit its log records.

### Logging activation level

You can change the ldap3 logging activation level to a different one if you need not to mix logging from ldap3 with DEBUG level for your application:

```
import logging
logging.basicConfig(filename='client_application.log', level=logging.CRITICAL)
from ldap3.utils.log import set_library_log_activation_level
set_library_log_activation_level(logging.CRITICAL)  # ldap3 will emit its log only␣
↪when you set level=logging.CRITICAL in your log configuration
```

ldap3 logging has its own level of log detail: OFF, ERROR, BASIC, PROTOCOL, NETWORK and EXTENDED. You can set the level of detail with ldap3.utils.log.set_library_log_detail_level().

Detail level defaults to OFF. You must change it at runtime as needed to have anything logged:

```
from ldap3.utils.log import set_library_log_detail_level, OFF, BASIC, NETWORK,␣
↪EXTENDED
# ... unlogged ldap3 operation
set_library_log_detail_level(BASIC)
# ... ldap3 operation with few details
set_library_log_detail_level(EXTENDED)
# ... other ldap3 operation with most details
set_library_log_detail_level(OFF)
# nothing else is logged
```

## Logging detail level

Each detail level details a specific feature of the library and includes the previous level details, as for standard logging:

- OFF: nothing is logged
- ERROR: only exceptions are logged
- BASIC: library activity is logged, only operation result is shown
- PROTOCOL: LDAPv3 operations are logged, sent requests and received responses are shown
- NETWORK: socket activity is logged
- EXTENDED: ldap messages are decoded and properly printed

At EXTENDED level every LDAP message is logged and printed in a proper way (thanks to pyasn1 prettyPrint feature). The flow of the network conversation can be easily guessed by the prefix of the message lines: >> for outgoing messages (to the LDAP server) and << for incoming messages (from the LDAP server). To get a full descriptive logging of outgoing messages you must set `fast_decoder=False` in the connection object.

Each log record contains the detail level and when available information on the active connection used. So the log size grows very easily. ldap3 performance degrades when logging is active, especially at level greater than ERROR, so it's better to use it only when needed.

logging text is encoded to ASCII.

## Hiding sensitive data

Sensitive data (as user password and SASL credentials) are stripped by default from the log and substituted with a string of '*' (with the same length of the original value) or by a "<stripped xxx characters of sensitive data>" message (where xxx is the number of characters stripped). You can change the default behaviour and let the log record all the data with the set_library_log_hide_sensitive_data(False) function of the utils.log package:

```
import logging
logging.basicConfig(filename=log_file, level=logging.DEBUG)
from ldap3.utils.log import set_library_log_detail_level, get_detail_level_name, set_
↪library_log_hide_sensitive_data, EXTENDED

set_library_log_detail_level(EXTENDED)
set_library_log_hide_sensitive_data(False)
```

You can use the get_library_log_hide_sensitive_data() function of the utils.log module to check if sensitive data will be hidden or not.

## Maximum log line length

When logging LDAP responses at the EXTENDED detail level is possible to receive very long lines. This usually happens when the schema is read with get_info=SCHEMA or get_info=ALL in the Server object. A tipical response while reading the schema can be 300 KB (or more) long.

To avoid the creation of huge and usually useless logs the ldap3 library log system set the maximum length of the log lines to a default value of 4096 characters. This should be a reasonable value for logging search operation responses, without cluttering the log. If you want to change the maximum log line length to another value you can use the set_library_log_max_line_length(length) function to set the desired length. You can use the get_library_log_max_line_length() function to read the current value.

## Examples

Opening an SSL connection to an LDAP server listening on IPv4 only on a IPv6/IPv4 box. The connection mode is set to IP_V6_PREFERRED, the connection is bound and a search operation is performed

a search operation at basic level:

```
INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to BASIC
DEBUG:ldap3:BASIC:instantiated Tls: <Tls(validate=0)>
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=636, use_
↪ssl=True, tls=Tls(validate=0), get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=636, use_ssl=True, tls=Tls(validate=0), get_info='NO_INFO'), user='cn=admin,
↪o=test', password='********', auto_bind='NONE', version=3, authentication='SIMPLE',␣
↪client_strategy='SYNC', auto_referrals=True, check_names=True, collect_usage=True,␣
↪read_only=False, lazy=False, raise_exceptions=False)>
DEBUG:ldap3:BASIC:start BIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it> for <ldaps://openldap:636 -␣
↪ssl - user: cn=admin,o=test - unbound - closed - <local: [::]:50122 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
```

```
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 636)]
DEBUG:ldap3:BASIC:refreshing server info for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50123 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>
DEBUG:ldap3:BASIC:start SEARCH operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50123 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done SEARCH operation, result <True>
DEBUG:ldap3:BASIC:start UNBIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50123 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

the same operation at PROTOCOL detail level:

```
INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to PROTOCOL
DEBUG:ldap3:BASIC:instantiated Tls: <Tls(validate=0)>
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=636, use_
↪ssl=True, tls=Tls(validate=0), get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=636, use_ssl=True, tls=Tls(validate=0), get_info='NO_INFO'), user='cn=admin,
↪o=test', password='********', auto_bind='NONE', version=3, authentication='SIMPLE',␣
↪client_strategy='SYNC', auto_referrals=True, check_names=True, collect_usage=True,␣
↪read_only=False, lazy=False, raise_exceptions=False)>
DEBUG:ldap3:BASIC:start BIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it.> for <ldaps://openldap:636 -␣
↪ssl - user: cn=admin,o=test - unbound - closed - <local: [::]:50127 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 636)]
```

```
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50128 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'version': 3, 'authentication': {'sasl':␣
↪None, 'simple': '<stripped 8 characters of sensitive data>'}, 'name': 'cn=admin,
↪o=test'}> sent via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:50128 - remote: 192.168.137.104:636> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:PROTOCOL:BIND response <{'result': 0, 'saslCreds': None, 'type':
↪'bindResponse', 'message': '', 'referrals': None, 'dn': '', 'description': 'success
↪'}> received via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:50128 - remote: 192.168.137.104:636> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50128 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>
DEBUG:ldap3:BASIC:start SEARCH operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50128 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:SEARCH request <{'sizeLimit': 0, 'scope': 2, 'timeLimit': 0,
↪'typesOnly': False, 'filter': '(cn=test*)', 'attributes': ['objectClass', 'sn'],
↪'base': 'o=test', 'dereferenceAlias': 3}> sent via <ldaps://openldap:636 - ssl -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50128 - remote: 192.
↪168.137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:PROTOCOL:SEARCH response entry <{'type': 'searchResEntry', 'dn':
↪'cn=testSASL,o=test', 'attributes': {'objectClass': ['inetOrgPerson',
↪'organizationalPerson', 'person', 'top'], 'sn': ['testSASL']}, 'raw_attributes': {
↪'objectClass': [b'inetOrgPerson', b'organizationalPerson', b'person', b'top'], 'sn
↪': [b'testSASL']}}> received via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - bound - open - <local: 192.168.137.1:50128 - remote: 192.168.137.104:636> -
↪ tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done SEARCH operation, result <True>
DEBUG:ldap3:BASIC:start UNBIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50128 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50128 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

the same opeaton at NETWORK detail level:

```
INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to NETWORK
DEBUG:ldap3:BASIC:instantiated Tls: <Tls(validate=0)>
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=636, use_
↪ssl=True, tls=Tls(validate=0), get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ No strategy - async - real DSA - not pooled - cannot stream output>
```

```
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=636, use_ssl=True, tls=Tls(validate=0), get_info='NO_INFO'), user='cn=admin,
↪o=test', password='********', auto_bind='NONE', version=3, authentication='SIMPLE',␣
↪client_strategy='SYNC', auto_referrals=True, check_names=True, collect_usage=True,␣
↪read_only=False, lazy=False, raise_exceptions=False)>
DEBUG:ldap3:BASIC:start BIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:NETWORK:opening connection for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be␣
↪made because the target machine actively refused it> for <ldaps://openldap:636 -␣
↪ssl - user: cn=admin,o=test - unbound - closed - <local: [::]:50130 - remote:␣
↪[None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 636)]
DEBUG:ldap3:NETWORK:socket wrapped with SSL using SSLContext for <ldaps://
↪openldap:636 - ssl - user: cn=admin,o=test - unbound - closed - <local: [None]:None␣
↪- remote: [None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection open for <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - unbound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.104:636>
↪ - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50131 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'version': 3, 'authentication': {'sasl':␣
↪None, 'simple': '<stripped 8 characters of sensitive data>'}, 'name': 'cn=admin,
↪o=test'}> sent via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound -␣
↪open - <local: 192.168.137.1:50131 - remote: 192.168.137.104:636> - tls not started␣
↪- listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50131 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - unbound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.104:636>
↪ - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50131 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50131 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:PROTOCOL:BIND response <{'description': 'success', 'referrals': None,
↪'result': 0, 'type': 'bindResponse', 'message': '', 'saslCreds': None, 'dn': ''}>␣
↪received via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound - open -
↪<local: 192.168.137.1:50131 - remote: 192.168.137.104:636> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>
DEBUG:ldap3:BASIC:start SEARCH operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:SEARCH request <{'attributes': ['objectClass', 'sn'], 'base':
↪'o=test', 'scope': 2, 'dereferenceAlias': 3, 'filter': '(cn=test*)', 'typesOnly':␣
↪False, 'sizeLimit': 0, 'timeLimit': 0}> sent via <ldaps://openldap:636 - ssl -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.
↪168.137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:sent 63 bytes via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.104:636> -
↪ tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 114 bytes via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:SEARCH response entry <{'raw_attributes': {'sn': [b'testSASL'],
↪'objectClass': [b'inetOrgPerson', b'organizationalPerson', b'person', b'top']},
↪'attributes': {'sn': ['testSASL'], 'objectClass': ['inetOrgPerson',
↪'organizationalPerson', 'person', 'top']}, 'type': 'searchResEntry', 'dn':
↪'cn=testSASL,o=test'}> received via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.104:636> -
↪ tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done SEARCH operation, result <True>
DEBUG:ldap3:BASIC:start UNBIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.104:636> -
↪ tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50131 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:NETWORK:connection closed for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - closed - <no socket> - tls not started - not listening -␣
↪SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

the same operation at EXTENDED detail level:

```
INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG -␣
↪available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED -␣
↪sensitive data will be hidden
DEBUG:ldap3:ERROR:detail level set to EXTENDED
DEBUG:ldap3:BASIC:instantiated Tls: <Tls(validate=0)>
DEBUG:ldap3:BASIC:instantiated Server: <Server(host='openldap', port=636, use_
↪ssl=True, tls=Tls(validate=0), get_info='NO_INFO')>
DEBUG:ldap3:BASIC:instantiated Usage object
DEBUG:ldap3:BASIC:instantiated <SyncStrategy>: <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ No strategy - async - real DSA - not pooled - cannot stream output>
DEBUG:ldap3:BASIC:instantiated Connection: <Connection(server=Server(host='openldap',␣
↪port=636, use_ssl=True, tls=Tls(validate=0), get_info='NO_INFO'), user='cn=admin,
↪o=test', password='********', auto_bind='NONE', version=3, authentication='SIMPLE',␣
↪client_strategy='SYNC', auto_referrals=True, check_names=True, collect_usage=True,␣
↪read_only=False, lazy=False, raise_exceptions=False)>
DEBUG:ldap3:BASIC:start BIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:NETWORK:opening connection for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - closed - <no socket> - tls not started - not listening -
↪ SyncStrategy>
DEBUG:ldap3:BASIC:reset usage metrics
DEBUG:ldap3:BASIC:start collecting usage metrics
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]>
DEBUG:ldap3:BASIC:address for <ldaps://openldap:636 - ssl> resolved as <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]>
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET6: 23>, <SocketKind.SOCK_STREAM: 1>, 6, '', (
↪'fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:obtained candidate address for <ldaps://openldap:636 - ssl>: <[
↪<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104',␣
↪636)]> with mode IP_V6_PREFERRED
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET6: 23>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('fe80::215:5dff:fe8f:2f0d%20', 636, 0, 20)]
DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] Impossibile stabilire la␣
↪connessione. Rifiuto persistente del computer di destinazione> for <ldaps://
↪openldap:636 - ssl - user: cn=admin,o=test - unbound - closed - <local: [::]:50132 -
↪ remote: [None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:BASIC:try to open candidate address [<AddressFamily.AF_INET: 2>,
↪<SocketKind.SOCK_STREAM: 1>, 6, '', ('192.168.137.104', 636)]
DEBUG:ldap3:NETWORK:socket wrapped with SSL using SSLContext for <ldaps://
↪openldap:636 - ssl - user: cn=admin,o=test - unbound - closed - <local: [None]:None␣
↪- remote: [None]:None> - tls not started - not listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection open for <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.104:636>
↪ - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:PROTOCOL:performing simple BIND for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:simple BIND request <{'authentication': {'sasl': None, 'simple':
↪'<stripped 8 characters of sensitive data>'}, 'name': 'cn=admin,o=test', 'version':␣
↪3}> sent via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound - open -
↪<local: 192.168.137.1:50133 - remote: 192.168.137.104:636> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <1> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=1
>> protocolOp=ProtocolOp:
>>  bindRequest=BindRequest:
>>   version=3
>>   name=b'cn=admin,o=test'
>>   authentication=AuthenticationChoice:
>>    simple=<stripped 8 characters of sensitive data>
DEBUG:ldap3:NETWORK:sent 37 bytes via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.104:636>
↪ - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 14 bytes via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - unbound - open - <local: 192.168.137.1:50133 - remote: 192.168.
↪137.104:636> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=1
<< protocolOp=ProtocolOp:
<<  bindResponse=BindResponse:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:BIND response <{'dn': '', 'description': 'success', 'type':
↪'bindResponse', 'message': '', 'result': 0, 'saslCreds': None, 'referrals': None}>␣
↪received via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound - open -
↪<local: 192.168.137.1:50133 - remote: 192.168.137.104:636> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:refreshing server info for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:BASIC:done BIND operation, result <True>
DEBUG:ldap3:BASIC:start SEARCH operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:SEARCH request <{'scope': 2, 'base': 'o=test', 'timeLimit': 0,
↪'filter': '(cn=test*)', 'typesOnly': False, 'attributes': ['objectClass', 'sn'],
↪'dereferenceAlias': 3, 'sizeLimit': 0}> sent via <ldaps://openldap:636 - ssl -␣
↪user: cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.
↪168.137.104:636> - tls not started - listening - SyncStrategy>
```

```
DEBUG:ldap3:PROTOCOL:new message id <2> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sent via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=2
>> protocolOp=ProtocolOp:
>>  searchRequest=SearchRequest:
>>   baseObject=b'o=test'
>>   scope='wholeSubtree'
>>   derefAliases='derefAlways'
>>   sizeLimit=0
>>   timeLimit=0
>>   typesOnly='False'
>>   filter=Filter:
>>    substringFilter=SubstringFilter:
>>     type=b'cn'
>>     substrings=Substrings:
>>      Substring:
>>       initial=b'test'
>>   attributes=AttributeSelection:
>>    b'objectClass'    b'sn'
DEBUG:ldap3:NETWORK:sent 63 bytes via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.104:636> -
↪ tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 114 bytes via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>:
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  searchResEntry=SearchResultEntry:
<<   object=b'cn=testSASL,o=test'
<<   attributes=PartialAttributeList:
<<    PartialAttribute:
<<     type=b'sn'
<<     vals=Vals:
<<      b'testSASL'
<<    PartialAttribute:
<<     type=b'objectClass'
<<     vals=Vals:
<<      b'inetOrgPerson'      b'organizationalPerson'      b'person'      b'top'
DEBUG:ldap3:NETWORK:received 14 bytes via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:received 1 ldap messages via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message received via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>:
```

```
<<LDAPMessage:
<< messageID=2
<< protocolOp=ProtocolOp:
<<  searchResDone=SearchResultDone:
<<   resultCode='success'
<<   matchedDN=b''
<<   diagnosticMessage=b''
DEBUG:ldap3:PROTOCOL:SEARCH response entry <{'attributes': {'sn': ['testSASL'],
↪'objectClass': ['inetOrgPerson', 'organizationalPerson', 'person', 'top']}, 'dn':
↪'cn=testSASL,o=test', 'type': 'searchResEntry', 'raw_attributes': {'sn': [b'testSASL
↪'], 'objectClass': [b'inetOrgPerson', b'organizationalPerson', b'person', b'top']}}>
↪ received via <ldaps://openldap:636 - ssl - user: cn=admin,o=test - bound - open -
↪<local: 192.168.137.1:50133 - remote: 192.168.137.104:636> - tls not started -␣
↪listening - SyncStrategy>
DEBUG:ldap3:BASIC:done SEARCH operation, result <True>
DEBUG:ldap3:BASIC:start UNBIND operation via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:UNBIND request sent via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:PROTOCOL:new message id <3> generated
DEBUG:ldap3:NETWORK:sending 1 ldap message for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:EXTENDED:ldap message sending via <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>:
>>LDAPMessage:
>> messageID=3
>> protocolOp=ProtocolOp:
>>  unbindRequest=b''
DEBUG:ldap3:NETWORK:sent 7 bytes via <ldaps://openldap:636 - ssl - user: cn=admin,
↪o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.104:636> -
↪ tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:closing connection for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - open - <local: 192.168.137.1:50133 - remote: 192.168.137.
↪104:636> - tls not started - listening - SyncStrategy>
DEBUG:ldap3:NETWORK:connection closed for <ldaps://openldap:636 - ssl - user:␣
↪cn=admin,o=test - bound - closed - <no socket> - tls not started - not listening -␣
↪SyncStrategy>
DEBUG:ldap3:BASIC:stop collecting usage metrics
DEBUG:ldap3:BASIC:done UNBIND operation, result <True>
```

At the ERROR detail level you get only the library errors:

> INFO:ldap3:ldap3 library initialized - logging emitted with loglevel set to DEBUG - available detail levels are: OFF, ERROR, BASIC, PROTOCOL, NETWORK, EXTENDED - sensitive data will be hidden DEBUG:ldap3:ERROR:detail level set to ERROR DEBUG:ldap3:ERROR:<socket connection error: [WinError 10061] No connection could be made because the target machine actively refused it.> for <ldaps://openldap:636 - ssl - user: cn=admin,o=test - unbound - closed - <local: [::]:50321 - remote: [None]:None> - tls not started - not listening - SyncStrategy>

The usage metrics are the same at every detail:

> **Connection Usage:**
>
> > **Time: [elapsed: 0:00:01.949587]** Initial start time: 2015-05-18T19:27:17.057422 Open socket time: 2015-05-18T19:27:17.057422 Close socket time: 2015-05-18T19:27:19.007009

**Server:** Servers from pool: 0 Sockets open: 1 Sockets closed: 1 Sockets wrapped: 1

**Bytes: 249** Transmitted: 107 Received: 142

**Messages: 6** Transmitted: 3 Received: 3

**Operations: 3** Abandon: 0 Bind: 1 Add 0 Compare: 0 Delete: 0 Extended: 0 Modify: 0 ModifyDn: 0 Search: 1 Unbind: 1

**Referrals:** Received: 0 Followed: 0

**Restartable tries: 0** Failed restarts: 0 Successful restarts: 0

## Mocking

You can mock ldap3 in your project using the MockSyncStrategy or the MockAsyncStrategy. Both of these strategies are based on the MockBaseStreategy that emulates a simple LDAP server and can be used to test the LDAP functionalities in your project. You can even define a specific kind of LDAP server to emulate and MockBaseStrategy will provide a suitable schema and the relevant DSA info.

To populate the DIT (the Directory Information Tree, the hierarchical database that contains the LDAP data) you can provide data from an actual LDAP server via a JSON data file or you can create at runtime only the entries and the attributes needed for your test suite.

Anyway if you want to use Simple Bind authentication you must provide users and passwords needed by the test. No ACL or rights control is done by the MockBaseStrategy.

To mock the ldap3 library in your project you must define a fake Server object and set the client_strategy attribute to MOCK_SYNC or MOCK_ASYNC while defining the Connection object:

```
from ldap3 import Server, Connection
server = Server('my_fake_server')
connection = Connection(server, user='cn=my_user,ou=test,o=lab', password='my_password
↪', client_strategy=MOCK_SYNC)
```

then you can load the json entries file to add data to the DIT:

```
connection.strategy.entries_from_json('my_entries.json')
```

or add entries dynamically at runtime:

```
connection.strategy.add_entry('cn=user0,ou=test,o=lab', {'userPassword': 'test0000',
↪'sn': 'user0_sn', 'revision': 0})
connection.strategy.add_entry('cn=user1,ou=test,o=lab', {'userPassword': 'test1111',
↪'sn': 'user1_sn', 'revision': 0})
connection.strategy.add_entry('cn=user2,ou=test,o=lab', {'userPassword': 'test2222',
↪'sn': 'user2_sn', 'revision': 0})
```

**Note:** MockBaseStrategy doesn't check the validity of the added entries against the schema, so you can just add the entries and the attribute values needed to perform your tests.

Then you can use the mock connection as a normal connection to an LDAP server.

**Note:** The MockBaseStrategy provides only Simple Authentication bind. You can bind to any object in the dict that has a **userPassword** attribute (either single or multi-valued). The password must be stored as cleartext. You cannot

use the `auto_bind` parameter because the DIT is populated after the creation of the Connection object.

MockBaseStrategy supports the Bind, Unbind, Add, Modify, ModifyDn, Compare, Delete and Search operations (except for the extensible match). Abandon and Extended are not supported.

**Note:** You can replicate the DIT of a real server (or just the portions of the Tree that you need in your tests) using the `response_to_file()` method of the Connection object with *raw* output. Just perform a SUBTREE search with ALL_ATTRIBUTES (or the attributes needed in your tests) with the needed base and a filter as `(objectclass=*)` that captures every object in the DIT:

```
from ldap3 import Server, Connection, ALL_ATTRIBUTES
server = Server('my_real_server')
connection = Connection(server, 'cn=my_real_user,ou=test,o=lab', 'my_real_password',␣
↪auto_bind=True)
if connection.search('ou=test,o=lab', '(objectclass=*)', attributes=ALL_ATTRIBUTES):
    connection.response_to_file('my_entries.json', raw=True)
```

The *my_entries.json* file can then be used in the `entries_from_json()` method of the MockBaseStrategy.

While defining the mock server you can specify a predefined schema with the `get_info` parameter:

```
from ldap3 import Server, Connection, OFFLINE_SLAPD_2_4
server = Server('my_fake_server', get_info=OFFLINE_SLAPD_2_4)
```

The available offline schemas are OFFLINE_SLAPD_2_4 (OpenLDAP), OFFLINE_EDIR_8_8_8 (eDirectory), OFFLINE_AD_2012_R2 (Active Directory) and OFFLINE_DS389_1_3_3 (389 Directory Server).

You can also speficy a previously saved schema and info retrieved from a real server:

```
from ldap3 import Server, Connection, ALL

# Retrieve server info and schema from a real server
server = Server('my_real_server', get_info=ALL)
connection = Connection(server, 'cn=my_user,ou=test,o=lab', 'my_real_password', auto_
↪bind=True)

# Store server info and schema to json files
server.info.to_file('my_real_server_info.json')
server.schema.to_file('my_real_server_schema.json')
```

## A complete example

The following code retrieves the schema and the server info from a real server, then read the entries from a portion of the DIT and store them in a json file. Then a fake server is created and loaded with the previoulsy saved schema, server info and entries and a fake user is defined for simple binding:

```
from ldap3 import Server, Connection, ALL, ALL_ATTRIBUTES
REAL_SERVER = 'my_real_server'
REAL_USER = 'cn=my_real_user,ou=test,o=lab'
REAL_PASSWORD = 'my_real_password'

# Retrieve server info and schema from a real server
server = Server(REAL_SERVER, get_info=ALL)
connection = Connection(server, REAL_USER, REAL_PASSWORD, auto_bind=True)
```

```
# Store server info and schema to json files
server.info.to_file('my_real_server_info.json')
server.schema.to_file('my_real_server_schema.json')

# Read entries from a portion of the DIT from real server and store them in a json␣
↪file
if connection.search('ou=test,o=lab', '(objectclass=*)', attributes=ALL_ATTRIBUTES):
    connection.response_to_file('my_real_server_entries.json', raw=True)

# Close the connection to the real server
connection.unbind()

# Create a fake server from the info and schema json files
fake_server = Server.from_definition('my_fake_server', 'my_real_server_info.json',
↪'my_real_server_schema.json')

# Create a MockSyncStrategy connection to the fake server
fake_connection = Connection(fake_server, user='cn=my_user,ou=test,o=lab', password=
↪'my_password', client_strategy=MOCK_SYNC)

# Populate the DIT of the fake server
fake_connection.strategy.entries_from_json('my_real_server_entries.json')

# Add a fake user for Simple binding
fake_connection.strategy.add_entry('cn=my_user,ou=test,o=lab', {'userPassword': 'my_
↪password', 'sn': 'user_sn', 'revision': 0})

# Bind to the fake server
fake_connection.bind()
```

Then the connection is ready to be used in your tests.

## Testing

Inside the "test" package you can find examples for each LDAP operation. You can customize the test modifying the variables in the __init__.py in the test package. You can set the following parameters:

```
test_server = 'server'  # the LDAP server where tests are executed
test_user = 'user'  # the user that performs the tests
test_password = 'password'  # user's password
test_base = 'o=test'  # base context where test objects are created
test_moved = 'ou=moved,o=test'  # base context where objects are moved in ModifyDN␣
↪operations
test_name_attr = 'cn'  # naming attribute for test objects
test_port = 389  # ldap port
test_port_ssl = 636  # ldap secure port
test_authentication = SIMPLE  # authentication type
test_strategy = SYNC  # strategy for executing tests
#test_strategy = ASYNC  # uncomment this line to test the async strategy
# test_strategy = RESTARTABLE  # uncomment this line to test the sync_restartable␣
↪strategy
```

To execute the test suite you need an LDAP server with the test_base and test_moved containers and a test_user with privileges to add, modify and remove objects in that context.

To execute the test_tls unit test you must supply your own certificates or tests will fail.

The test package is available in the git repository.

# CHANGELOG

**# 2.4 - not yet released**

- security fix in the rebind() method of the Connection object (thanks Daniel)
- fix for Sasl credentials in Python 3 (thanks Busuwe)
- fixed bug when checking for equality in MockBase
- added validator parameter to Server object for custom validators
- attribute values are now validated in add/compare/modify operations in the Connection object
- Python types can now be used in add/compare/modify operations
- compatible with the pyasn1 library from version 0.1.8 up to latest (0.3.3 for now) version
- fixed compatibility with Twisted on Windows on Python 2.7 (thanks Pmisik)
- fixed paged_search behaviour in Reader object
- fixed regression in MockBase (thanks Markus)

**# 2.3 - 2017.08.02**

- compatible with the pyasn1 library from version 0.1.8 up to latest (0.3.1 for now) version
- MockAsync strategy is available
- added __ne__ method to Attribute in abstraction layer (thank Rodrigo)
- added LDAPUserNameIsMandatoryError exception in simple bind when user name is empty
- search referrals are properly decoded with fast decoder
- paged search works in mock strategies
- paged_search in extend.standard namespace raises an exception of class LDAPOperationResult if the search returns an error
- search_paged() method of Cursor object now return the whole list of entries if generator=False
- updated docs for defaults parameters (thanks Guarnacciaa)
- fixed mockBase for integer matching (thanks Jijo)
- boolean values are now uppercase in LDIF (thanks Linus)
- fixed timeout in ssl connection on Linux and Mac (thanks Allan)
- changed some internal functions to private in ldap3.utils.dn
- operational attribute entryDN is properly managed in Mock strategies (thanks Mark)
- new rdn in renamed entry is properly set in Mock strategies (thanks Mark)
- metrics are now updated for Mock strategies, except that for received bytes (thanks joehy)
- better managing of missing schema from the server (thanks Deborah)
- fixed error while schema is not in string format (thanks Alexandre)
- SNI support added when the underlying python library allows it (thanks Edmund)

- added pool_keepalive parameter to Connection object for REUSABLE strategy
- connection.extend.microsoft.modify_password returns False when change is not successful (thanks Ashley)
- added validators for uuid and uuid_le
- fixed error while searching for bytes
- fixed pickling and unpickling of datetime values (thanks David)
- fixed error that resulted in valid generalizedTime strings not being parsed (thanks Busuwe)
- fixed error with modify operation on referrals (thanks Busuwe)
- fixed error in mockBase add_entry() with raw rdn (thanks Chad)
- fixed error when stdin has not encoding in config.py (thanks cronicryo)
- fixed error when optional field are not present in pyasn1 requests (thanks Ilya)
- added DEFAULT_SERVER_ENCODING config parameter, should always be utf-8
- DEFAULT_ENCODING config parameter renamed to DEFAULT_CLIENT_ENCODING
- ADDITIONAL_ENCODINGS config parameter renamed to ADDITIONAL_SERVER_ENCONDINGS
- additional encodings are applied to all data received from the server
- additional encodings are not applied to client data
- added from_server=False parameter to to_unicode() to not try client encoding while decoding data from server

**# 2.2.4 - 2017.05.07**

- leading and trailing spaces in server name don't raise exception anymore
- DitContentRule is properly read from the schema
- added validator for Active Directory timestamp
- Mock strategies raise an exception if a non-bytes value is added to the schema when no offline schema is provided (str and int are automatically converted)
- added custom_validators property to Mock strategies
- modifying objectClass with bytes values doesn't raise an exception anymore (but it may fail anyway because of server constraints)
- ensure that config sequence parameters are properly set
- allow case insensitive attribute and class names in config parameters
- added server.schema.is_valid() to check if the schema is available
- empty schema properties are set to empty dict() instead of None
- schema definitions with trailing and leading spaces are now properly parsed and don't raise an LDAPSchemaError exception anymore
- fixed error when flaky servers (OpenLDAP) don't return the correct response with StartTls

**# 2.2.3 - 2017.04.30**

- abstraction layer query converts int values to string (thanks dgadmin)
- CaseInsensitiveDictWithAlias doesn't raise an exception anymore if alias is set multiple times to the same key

- friendly names in AttrDef are properly managed when performing commits in Writer cursors
- no more errors when server returns an empty schema (thanks Glen)
- range attributes in entries are properly recognized when auto_range is False
- fixed random errors in auto_range searches (thanks James)
- fixed checking of malformed schema
- added configuration parameter IGNORE_MALFORMED_SCHEMA to not raise exception for servers that don't follow the LDAP RFCs (defaults to False)
- test config moved to test/config.py
- testcase_id generated randomly for each test
- added ATTRIBUTES_EXCLUDED_FROM_OBJECT_DEF parmeter to exclude some attribute from automatic populate of ObjectDef in Abstract Layer (helpful for AD)
- added IGNORED_MANDATORY_ATTRIBUTES_IN_OBJECT_DEF parmeter to exclude some attribute from mandatory attribute list in ObjectDef in Abstract Layer (helpful for AD)
- fixed error when using implicit assigning in WritableEntry
- added LDAPInvalidValueError Exception
- in Python 3 byte filter are converted to unicode before parsing
- RESPONSE_DN_ENCODING parameter renamed to ADDITIONAL_ENCODINGS
- to_unicode(value, encoding=None, additional_encodings=False) now checks for additional encoodings in ADDITIONAL_ENCODINGS list if additional_encoding is set to True
- Reusable strategy uses not lazy Restartable connections

. Reusable strategy doesn't keep requesting the schema - connection pool size in Reusable strategy defaults to 5
- optimized usage of configuration parameters

**# 2.2.2 - 2017.03.17**

- PLAIN mechanism added to SASL authentication (thanks Janusz)
- added RESULT_RESERVED return code (thanks Rak)
- added RESPONSE_DN_ENCODING in config for flaky servers that return non utf-8 encoded DN. Set it to a list of encoding to sequentially try for decodign DNs.
- removed StopIteration in generators (deprecated by PEP 479)
- fixed a bug when converting AD datetimes on Windows systems
- added compatibility with pyasn1 0.2.3
- fixed NTLM authentication with pyasn1 0.2.3
- fixed an error when installing via executable on Windows (thanks TrumSteven)
- added 'raw_dn' key in search response dictionary. It contains the DN byte value returned for DN by the server
- attributes with ";binary" tag can now be retrieved in abstraction layer with the usual entry.atttribute syntax
- updated tests for OpenLDAP
- fixed error when in add/remove extend operation for case mismatch in user or group dn
- integer validator now automatically convert valid string numbers to int

- invalid timezone are checked when validating Generalized Time Format
- added test cases for validators
- updated tests for OpenLDAP

**# 2.2.1 - 2017.02.12**

- tested against pyasn1 0.2.2 (thanks Ilya)
- get_response() has an optional new parameter "get_request" to return the request too, helpful in asynchronous strategies
- connection.request, connection.response and connection result are now properly blanked in async strategies
- ldap3.utils.dn.safe_dn() now checks for AD names only if no equal sign is present in the dn
- abstraction layer properly works with asynchronous strategies
- added a named tuple "Operation" used to store the request, result and response of an LDAP operation in Cursor history
- cursors in the Abstraction Layer keep history of executed LDAP operations for the last Cursor operation in the cursor.operation property
- Cursors in the Abstraction Layer keep history of errors for the last Cursor operation in the cursor.errors property
- if any error has occurred in the last operation of a Cursor the cursor.failed property is set to True
- added a named tuple "Operation" for storing request, result and response of an LDAP operation in Cursor history
- Cursor honours raise_exception parameter of the Connection.
- Cursor commit() return True if operations are successful and False if not. All pending commits are executed even if some fail
- new entries that have no additional mandatory attributes other those defined in dn are properly managed in Writers (thanks Matt)
- CaseInsensitiveDict now properly strips blanks from keys
- updated hashing alghoritm SHA to SHA1 (thanks Satoh)
- added match_dn(dn) to Cursor for matching entries with specified text in DN
- added match(attributes, value) for matching entries with specified value in one or more attribute values. It checks values and raw_values
- Cursors have simple match capability. When key is a string Cursor tries to match it against the DN of entries found.

**# 2.2.0 - 2017.01.16**

- tested againsts Python 3.6.0, Python 2.7.13 and Python 2.6.6
- updated docs regarding search response attributes (thanks James)
- fixed LDIF representation for operation_to_ldif (thanks m7four)
- fixed rebind for pooled connections
- fixed custom sort order in LDIF representation of entry
- added Active Directory GUID syntax for safe_dn() (thanks dinhngtu)

- added pre-post read control (thanks Elizabeth)
- added add_members_to_groups in microsoft.extend namespace for Active Directory
- added remove_members_to_groups in microsoft.extend namespace for Active Directory
- refactored internal extend.microsoft and extend.novell structures
- fixed auto_escape for extended characters (thanks asand3r)
- validators now transform the Python value to a valid LDAP value when appropriate (thanks Sjd-Risca)
- added validator for boolean types
- added validator for date types
- fixed representation of binary data in Abstraction Layer for Python 2
- added auto_encode parameter to Connection object (defaults to True)
- limited auto_escape feature only to filter values
- escape_filter_chars doens't try anymore to guess if the value is already escaped.
- added ldap3.conv.is_filter_safe() (thanks Robert)
- added auto_escape parameter to connection.search() to override connection auto_escape behaviour (defaults to None)
- auto_escape is not applied to filter value if already escaped
- automatically encode output to stdout encoding for repr() and str() (for printing and logging attributes values).
- binary data are converted to a hex values string in repr() and str() (for printing and logging attributes values).
- auto_encoding is performed only for well known attribute types that use Unicode format in LDAP
- CLASSES_EXCLUDED_FROM_CHECK and ATTRIBUTES_EXCLUDED_FROM_CHECK moved to ldap3.utils.config and made available via get_config_parameter()
- added UTF8_ENCODED_SYNTAXES in ldap3.config.utils and made available via get_config_parameter()
- added UTF8_ENCODED_TYPES in ldap3.config.utils and made available via get_config_parameter()
- config parameters made available only via get_config_parameters()
- removed to_bytes() and check_escape() from ldap3.utils.conv (ambiguous functions)
- added connection.request to MockSync (thanks Fabian)
- tags are properly managed in add, compare and modify requests (thanks guidow)
- in Mock strategies single-valued attributes are properly managed
- in Mock strategies attributes type names are properly managed
- implemented extended operation machinery in MockBase
- implemented WhoAmI [RFC4532] in Mock strategies
- implemented GetBindDn [NOVELL] in Mock strategies
- implemented operational attributes machinery in MockBase
- implemented entryDN [RFC5020] operational attribute in MockBase
- Sphinx updated to 1.5.1

**# 2.1.1 - 2016.11.18**

- Mock strategy uses case insensitive matching when appropriate
- fixed error when adding a virtual attribute in the abstract Entry object
- fixed error messages in Entry moving and renaming
- Reverted default connection strategy to SYNC (thanks Mauro)
- Fixed tutorials (thanks Mauro)
- Fixed checking of schema in ObjectDef (thanks Pierre)
- Fixed checking of stdin in config (thanks Oleg)
- fixed commit of entry with async strategies
- fixed reading of entries in async strategies
- added cipher argument to Tls (thanks Nicolas)
- fixed bug when using the abstraction layer with lazy connections
- fixed case matching while adding new entry in Writer cursor (thanks t0neg)
- disabled auto_escape for byte values
- fixed auto_escape for python 2
- fixed tutorials (thanks Ivano)

**# 2.1.0 - 2016.11.03**

- changed default Connection strategy from SYNC to RESTARTABLE
- enable automatic escaping of assertion values
- fixed decoding error with check_name=False
- added auto_escape parameter in connection, for trying automatic filter and attribute values escape
- fixed checking of schema in MockBase
- SASLBindInProgress doesn't raise an exception anymore with raise_exceptions=True
- standard formatters are applied in mocking strategies when serching for exact match

**# 2.0.9 - 2016.10.28**

- removed sanitization of DN in bind operation because some servers accept non standard DN for Simple Bind

**# 2.0.8 - 2016.10.28**

- included referral caching (thanks TWAC)

**# 2.0.7 - 2016.10.27**

- FIRST RELEASE OF LDAP3 V2
- changed signature of ldap3.abstract.Reader object
- removed search_size_limit(), search_time_limit() and search_types_only in the Reader cursor
- fixed SASL in progress error (thanks Styleex)
- fixed ALL_ATTRIBUTES in MOCK_SYNC strategy (thanks Belgarion)
- ncorrect attribute type error message now includes the name of the attribute (Thanks Andrej)

- relaxed dn checking for Active Directory UserPrincipalName
- relaxed dn checking for Active Directory SamAccountName
- added checking of attribute name in add, compare and search operations
- added checking of class name in add operation
- renamed exception LDAPTypeError to LDAPAttributeError
- in sync strategies LDAP operations populate the last_error attribute of the connection in case of not RESULT_SUCCESS
- connection.return_empty_attributes defaults to True
- escaped filter by default
- fixed escaping of filter
- add move and rename to abstraction layer entry
- ldap3 namespace decluttered
- RESULT_CODES moved to ldap3.core.results
- compatibility constants removed
- exceptions removed from ldap3 namespace, defined in ldap3.core.exceptions only
- ADDRESS_INFO_REFRESH_TIME is now configurable via set_config_parameter
- Operational attribute prefix set to '**OA_**'
- Allows cert and key file in the same file (thanks Jan-Philip)
- Removed logging info when logging is disabled (thanks Dan)
- Updated copyright notice
- Refactored abstraction layer with full support for CRUD (Create, Read, Update, Delete) abstract operations
- Added WritableEntry and WritableAttribute to abstraction layer
- Added standard validators for attribute types and syntaxes defined in the standard LDAP schema
- Added custom validators for attribute values
- Added update capability to abstraction layer
- Fixed typo in docs (thanks Gerardwx)
- Fixed Object and Attribute representation in schema (superior class not shown)
- ObjectDef automatically populates attributes from schema, following object_class hierarchy
- Added attributes parameter to search* methods of Cursor, so that only needed attributes are read even if attr_defs defines more
- Fixed connect_timeout not honored while wrapping socket in tls (thanks Kyle)
- Added 'set' to SEQUENCE_TYPES (thanks Christian)
- Entries returned by search are now writable via the abstraction layer
- LDAPReaderError exception renamed to LDAPCursorError
- auto_range parameter in Connection defaults to True (thanks Ashley)
- get_info defaults to SCHEMA while defining Server object

- Included ordereddict 1.1 (# Copyright (c) 2009 Raymond Hettinger) in ldap3.utils.ordDict for backporting OrderedDict in Python 2.6
- Added config parameter RESET_AVAILABILITY_TIMEOUT to reinsert invalid address in candidate_addresses while checking connection, defaults to 5 seconds
- Fixed inability to connect to a server if the connection starts when the server is unavailable and then it becomes available again
- All DNs are sanitized if connection.check_names is True
- LDAPControlsError exception renamed to LDAPControlError
- LDAPChangesError exception renamed to LDAPChangeError
- The following older constants in ldap3 have been removed, please use the suggested ones:
- AUTH_ANONYMOUS = ANONYMOUS
- AUTH_SIMPLE = SIMPLE
- AUTH_SASL = SASL
- SEARCH_SCOPE_BASE_OBJECT = BASE
- SEARCH_SCOPE_SINGLE_LEVEL = LEVEL
- SEARCH_SCOPE_WHOLE_SUBTREE = SUBTREE
- SEARCH_NEVER_DEREFERENCE_ALIASES = DEREF_NEVER
- SEARCH_DEREFERENCE_IN_SEARCHING = DEREF_SEARCH
- SEARCH_DEREFERENCE_FINDING_BASE_OBJECT = DEREF_BASE
- SEARCH_DEREFERENCE_ALWAYS = DEREF_ALWAYS
- STRATEGY_SYNC = SYNC
- STRATEGY_ASYNC_THREADED = ASYNC
- STRATEGY_LDIF_PRODUCER = LDIF
- STRATEGY_SYNC_RESTARTABLE = RESTARTABLE
- STRATEGY_REUSABLE_THREADED = REUSABLE
- STRATEGY_MOCK_SYNC = MOCK_SYNC
- STRATEGY_MOCK_ASYNC = MOCK_SYNC
- POOLING_STRATEGY_FIRST = FIRST
- POOLING_STRATEGY_ROUND_ROBIN = ROUND_ROBIN
- POOLING_STRATEGY_RANDOM = RANDOM
- GET_NO_INFO = NONE
- GET_DSA_INFO = DSA
- GET_SCHEMA_INFO = SCHEMA
- GET_ALL_INFO = ALL

**# 1.4.0 - 2016.07.18**

- Multiple Mock strategies now share entries when using the same Server object
- Added AsyncStreamStrategy

- Added Connection.extend.standard.persistent_search() (Thanks martinrm77)
- Added escaping of character > 0x7F in filter validation (thanks cfelder)
- Added better descriptions of Exception in abstraction layer (thanks cfelder)
- Added queue in Persistent Search
- Added callback in Persistent Search
- MockStrategy now honors raise_exception parameter (thanks Simon)

**# 1.3.3 - 2016.07.03**

- Change paameter name from 'check' to 'fix' in connection.extend.novell.add_members_to_groups() and connection.extend.novell.remove_members_from_groups
- Added connection.extend.novell.check_groups_memberships() that check if members are in groups and fixes the user-group relation if incorrect
- Updated docs link to ldap3.readthedocs.io
- Fixed error in utils.conv.check_escape (thanks Anjuta)
- Fixed typo in server.py when IP_V4_PREFERRED is used (thanks eva8668)
- Host name certificate matching exception and logging is much more informative (thanks eddie-dunn)
- Fixed typo in docs for use_ssl (thanks Brooks Kindle)
- Tested against Python 2.6., Python 2.7.12, Python 3.5.2 and PyPy 5.3.1

**# 1.3.2 - 2016.07.01**

- unreleased on pypi

**# 1.3.1 - 2016.05.11**

- Added support for mocking the ldap3 library
- Added support for MockSync strategy (thanks Roxana)
- Added checked_attributes=True parameter to connection.response_to_json()
- Added checked_attributes=True parameter to entry.entry_to_json()
- MockSyncBase strategy supports bind(), unbind(), delete(), compare(), modify(), modify_dn(), abandon(), add()
- MockSyncBase strategy accepts directory entries in json file
- Fixed schema representation (thanks Conrado)
- Allow connection.abandon(0), useful to "ping" the server
- Added connection.abandon() test suite
- Reusable strategy checks bind credential at bind() time, only on one worker connection
- Reusable strategy ignores abandon() operation because of multiple connection workers
- Reusable strategy honours return_empty_attributes connection parameter
- Added lazy information to connection representation
- Added support for hash (LM:NTLM) Windows NTLM authentication (thanks Dirk)
- Fixed representation of empty attributes in connection.entries
- Comparison of entry attributes value is easier

- Added new extended operation connection.extend.novell.start_transaction()
- Added new extended operation connection.extend.novell.end_transaction()
- Added new extended operation connection.extend.novell.add_members_to_groups(members, groups, check, transaction)
- Added new extended operation connection.extend.novell.remove_members_from_groups(members, groups, check, transaction)
- Added new exception LDAPTransactionError
- Added logic to handle Novell Transaction Error Unsolicited Notice
- Ignore cheching of ssl context when cadata, cafile and capath are not provided (thanks DelboyJan)

**# 1.2.2 - 2016.03.23**

- repr encoding set to 'ascii' when sys.stdout.encoding is None (thanks Jeff)

**# 1.2.1 - 2016.03.19**

- try to use the requested ssl protocol in SSLContext for Python>=3.4 (thanks Patrick)
- added return_empty_attributes to Connection object to return an empty list when the attribute requested is missing in the retrieved object

**# 1.1.2 - 2016.03.10**

- Added rebind() method to Connection object to rebind with a different user (thanks Lorenzo)
- Added Tests for rebind operation
- Start_tls honored in referrals
- Default ldaps port honored in referrals
- Additional connection parameters honored in referrals and in the restartable strategy
- Server connection timeout is honored while connecting, connection receive timeout while receiving
- Extended operations followed on referrals (thanks Pavel)
- Added receive_timeout parameter in Connection object to set socket in non-blocking mode with a specified timeout (thanks Antho)
- Fixed abstract entry __getattr__() throwing KeyError instead of AttributeError (thanks Kilroy)
- Fixed start_tls() Reusable strategy

**# 1.0.4 - 2016.01.25**

- Controls can be added to extended operation in the extend package (thanks Hinel)

**# 1.0.3 - 2015.12.1**

- Fixed set_config_parameter (thanks Sigmunau)
- Disabled unauthenticated authentication, see RFC 4513 section 5.1.2 (thanks Petros)
- Fixed falsey value in abstract Entry object __contains__() (thanks Vampouille)

**# 1.0.2 - 2015.12.07**

- Allowed_referral_hosts in Server objects defaults to [('*', True)] to accept any referral server with authentication
- Referral uri is now properly percent-undecoded (thanks TWAC)
- Referral Server object now use the same configuration of the original Server object

- Fixed __contains__() in Entry object (thanks Vampouille)

**# 1.0.1 - 2015.12.06**

- Removed the compat package
- Refactored docs for extend operations

**# 1.0.0 - 2015.12.06**

- Private RC for production
- Status moved to 5 - Production/Stable

**# 0.9.9.4 - 2015.12.02**

- Added items() to CaseInsensitiveDict class (thanks Jan-Hendrik)
- Added set_config_parameter() in ldap3 namespace to modify the values of the configurable parameters of ldap3
- Added microsoft.extend.modify_password() extended operation to change AD password
- Fixed find_active_random_server() in pooling (thanks Sargul)
- Fixed referral decoding in fast ber decoder (thanks TWAC)

**# 0.9.9.3 - 2015.11.15**

- Added LDAPI (LDAP over IPC) support for unix socket communication
- Added mandatory_in and optional_in in server schema for attribute types. Now you can see in which classes attributes are used
- Added last_transmitted_time and last_received_time to Usage object to track time of the last sent and received operation
- Exception SessionTerminatedByServer renamed to SessionTerminatedByServerError and added to ldap3 namespace
- Added get_config_parameter() in ldap3 namespace to read the current value of ldap3 configurable parameters
- Added SASL mechanism name as constants in the ldap3 namespace
- Added escape_filter_chars in utils.conv (thanks Peter)
- Reverted ALL_ATTRIBUTES behaviour in search to 0.9.9.1 (thanks Petros)

**# 0.9.9.2 - 2015.10.19**

- Fixed hasattr() behaviour for Entry object in Python 3
- Allows empty sasl_credentials in SASL bind
- Added POOLING_LOOP_TIMEOUT constant to specify how many seconds the server pooling strategy has to wait before retrying if it did not find an active server (defaults to 10)
- Pooling strategy now allows to specify the number of cycles to try when finding a server (with active=N)
- Pooling strategy now allows to specify how many seconds a server must be considered offline before retrying to check for availabiliry (with exhaust=N)
- Connection.entries defaults to empty list
- ALL_ATTRIBUTES don't send any attribute in the attribute list (was sending '*') while searching
- Added DirSync extended function for Microsoft Active Directory

- Added LDAP_SERVER_DIRSYNC_OID control for Microsoft Active Directory
- Added LDAP_SERVER_EXTENDED_DN_OID control for Microsoft Active Directory
- Added LDAP_SERVER_SHOW_DELETED_OID control for Microsoft Active Directory
- Fixed AD tests for single valued attributes
- Added ACL attribute in the ATTRIBUTES_EXCLUDED_FROM_CHECK list

**# 0.9.9.1 - 2015.09.21**

- Allows empty member values in groups while adding - this should not be as per rfc4511 4.1.7, but some servers expects it (thanks John)
- Faster case insensitive dict while getting and setting key (thanks Pierre)
- Updated setuptools to 18.3.2
- Updated wheel to 0.26
- Tested against Python 2.6 - Python 2.7 - Python 3.3 - Python 3.4 - Python 3.5 - pypy - pypy3

**# 0.9.9 - 2015.09.09**

- Fixed boolean value for True value in ASN.1 encoding for certain ldap servers. (thanks Will)
- Fixed follow auto referrals. (thanks WIll)
- Now protocol defined integer values can be used for scope and derefAliases arguments when searching. (thanks Will)
- Added description field in the AttrDef object. (thanks Hogne)
- Added a custom ber decoder. Decoding of received packets is now 10x faster.
- Added new boolean argument fast_decoder in connection object. Defaults to True.
- Highest date correctly managed by the format_ad_timestamp() formatter. (thanks Will)
- Fix for latest gssapi kerberos authentication module (thanks Alex)
- Added freeIPA OID descriptors
- Removed unneeded OidInfo class

**# 0.9.8.8 - 2015.08.14**

- Coerce objectClass to a list in Add operation. (thanks Yutaka)
- ObjectClass attribute values mantain their order in the Add operation. (thanks Yutaka)
- Fixed search filter composition when the value part of the assertion contains = character. (thanks Eero)
- Fixed modify_password extended operation when no hash method is specified. (thanks midnightlynx)
- Added credentials to kerberos authentication. (thanks Alex)
- Target name can be specified in sasl_credentials for Kerberos authentication. (thanks Alex)
- Target name can be read from DNS in sasl_credential for Kerberos authentication. (thanks Alex)
- Fixed connection.entries error when referrals are in the search response. (thanks WIll)

**# 0.9.8.7 - 2015.07.19**

- Backported ssl.match_hostname from Python 3.4.3 standard library to be used in Python < 2.7.10
- Use backports.ssl_match_hostname if present instead of static backported functions for matching server names in ssl certificate (thanks Michal)

- Attributes values are properly printed when not strings in abstract.attribute (thanks hogneh)
- Checking unicode __repr__() in python2
- Added hashing capability to Modify Password extended operation (thanks Gawain)

**# 0.9.8.6 - 2015.06.30**

- Modify operation now accept multiple changes for same attribute (Thanks Lorenzo)
- Fixed entries property in connection when objects from multiple object classes are returned
- Hide sensitive data in logging. use the utils.log.set_library_hide_sensitive_data(False) to show sensitive data and utils.log.get_library_hide_sensitive_data() to get the current value
- Limited number of characters in a single log line. use the utils.log.set_library_log_max_line_length(length) to set and utils.log.get_library_log_max_line_length(length) to get the current value
- Added CHANGES.txt with full changelog, latest changes only in README.txt

**# 0.9.8.5.post2 - 2015.06.24**

- Updated pyasn1 to 0.1.8
- Fixed error in not filter with pyasn1 0.1.8

**# 0.9.8.5 - 2015.06.23**

- Updated docs with ldap operations pages
- Fixed a bug where an Exception was raised on OpenBSD for missing IPV4_MAPPED flag
- Fixed missing add operation usage metrics
- Abstract Attribute doesn't permit "falsy" values or None as default (thanks Lucas)

**# 0.9.8.4 - 2015.05.19**

- Added EXTENDED log detail level with prettyPrint description of ldap messages
- Fixed logging of IPv6 address description
- Fixed checking of open address when dns returns more than one ip for the same host
- Fixed selection of proper address when failing back from IPv6 to IPv4 and vice-versa
- When sending controls controlValue is now optional (as stated in RFC 4511), specify None to not send it
- Moved badges to shields.io

**# 0.9.8.3 - 2015.05.11**

- Added support for logging
- Added LDAPInvalidTlsSpecificationError exception
- Added support for kerberos sasl - needs the gssapi package (thanks sigmaris and pefoley2)
- Added support for using generator objects in ldap operations (thanks Matt)
- Fixed bug in collect_usage (thanks Philippe)
- Changed default server mode from IP_SYSTEM_DEFAULT to IP_V6_PREFERRED

**# 0.9.8.2 - 2015.04.08**

- SaslCred returned as raw bytes (thanks Peter)
- Search_paged now properly works in abstract.reader (thanks wazboy)

**# 0.9.8.1 - 2015.04.04**

- Added NTLMv2 authentication method
- extend.standard.who_am_i() now try to decode the authzid as unicode
- Tests for AD (Active Directory) now use tls_before_bind when opening a connection
- 0.9.8 not working for pypi problems

**# 0.9.7.12 - 2015.03.18**

- Fixed missing optional authzid in digestMD5 sasl mechanism (thanks Damiano)
- Changed unneeded classmethods to staticmethods

**# 0.9.7.11 - 2015.03.12**

- Fixed address_info resolution on systems without the IPV4MAPPED flag (thanks Andryi)

**# 0.9.7.10 - 2015.02.28**

- Fixed bug in PagedSearch when server has a hard limit on the number of entries returned (thanks Reimar)
- 0.9.7.9 not working for pypi problems
- 0.9.7.8 not working for pypi problems
- 0.9.7.7 not working for pypi problems
- 0.9.7.6 not working for pypi problems

**# 0.9.7.5 - 2015.02.20**

- Fixed exception raised when opening a connection to a server. If there is only one candidate address and there is an error it returns the specific Exception, not a generic LDAPException error
- Address_info filters out any impossible address to reach
- Address_info include an IPV4MAPPED address for IPV6 host that try to reach an IPV4 only server
- Added SyncMock strategy (needs the sldap3 package)
- Fixed bug when using the aproximation operation in ldap search operations (thanks Laurent)
- Removed response from exception raised with raise_exceptions=True to avoid very long exceptions message

**# 0.9.7.4 - 2015.02.02**

- Added connection.entries property for storing response from search operations as and abstract.Entry collection.

**# 0.9.7.3 - 2015.01.25**

- Modify operation type can also be passed as integer

**# 0.9.7.2 - 2015.01.16**

- Fixed a bug when resolving IP address with getaddrinfo(). On OSX returned an UDP connection (thanks Hiroshi).

**# 0.9.7.1 - 2015.01.05**

- Moved to Github
- Moved to Travis-CI for continuous integration
- Moved to ReadTheDocs for documentation

- Moved testing servers in the cloud, to allow testing from Travis-CI
- Project renamed from python3-ldap to ldap3 to avoid name clashing with the existing python-ldap library
- Constant values in ldap3 are now strings. This is helpful in testing and debugging
- Test suite fully refactored to be used in cloud lab and local development lab
- Test suite includes options for testing against eDirectory, Active Directory and OpenLDAP

**# 0.9.7 - 2014.12.17**

- Fixed bug for auto_range used in paged search
- Added dual IP stack mode parameter in Server object, values are: IP_SYSTEM_DEFAULT, IP_V4_ONLY, IP_V4_PREFERRED, IP_V6_ONLY, IP_V6_PREFERRED
- Added read_server_info parameter to bind() and start_tls() to avoid multiple schema and info read operations with auto_bind
- Redesigned Reusable (pooled) strategy
- Added LDAPResponseTimeoutError exception raised when get_response() doesn't receive any response in the allowed timeout period
- Added shortened authentication parameters in ldap3 namespace: ANONYMOUS, SIMPLE, SASL
- Added shortened scope parameters in ldap3 namespace: BASE, LEVEL, SUBTREE
- Added shortened get_info parameters in ldap3 namespace: NONE, DSA, SCHEMA, ALL
- Added shortened alias dereferencing parameters in ldap3 namespace: DEREF_NONE, DEREF_SEARCH, DEREF_BASE, DEREF_ALWAYS
- Added shortened connection strategy parameters in ldap3 namespace: SYNC, ASYNC, LDIF, RESTARTABLE, REUSABLE
- Added shortened pooling strategy parameters in ldap3 namespace: FIRST, ROUND_ROBIN, RANDOM
- Added reentrant lock to avoid race conditions in the Connection object
- When runs in Python 2.7.9 uses SSLContext
- Tested against Python 2.7.9, PyPy 2.4.0 and PyPy3 2.4.0
- setuptools updated to 8.2.1

**# 0.9.6.2 - 2014.11.17**

- Changed SESSION_TERMINATED_BY_SERVER from 0 to -2
- Removed unneeded FORMAT_xxx variables in ldap3 namespace
- Fixed bug in auto_range when search operation returns search continuations
- Added infrastructure for Mock DSA (not functional yet)

**# 0.9.6.1 - 2014.11.11**

- Added boolean parameter "auto_range" to catch the "range" ldap tag in searches. When true all needed search operation are made to fully obtain the whole range of result values
- Fixed bug in sdist
- Added offline schema for Fedora 389 Directory Server 1.3.3
- Fixed bug while reading DSA info

**# 0.9.6 - 2014.11.01**

- New feature 'offline schema' to let the client have knowledge of schema and DSA info even if not returned by the server
- Offline schema for Novell eDirectory 8.8.8
- Offline schema for Microsoft Active Directory 2012 R2
- Offline schema for slapd 2.4 (Openldap)
- Added server.info.to_json() and server.info.to_file to JSON serialize schema and info from Server object
- Added Server.from_json() and Server.from_file() to create a Server object from a JSON definition
- Added response_to_json() and response_to_file() to Connection object to serialize search response entries in JSON as a string or as a file
- New exception hierarchy LDAPConfigurationError includes library configuration exceptions
- New exception LDAPInvalidConfigurationDefinitionError
- Dsa info and schema are not read twice when binding (thanks phobie)
- LDAPStartTLSError exception is merged with exception raised from ssl packaged
- Digest-MD5 SASL authentication accepts directives with list attributes (thanks John)
- Fixed caseInsensitiveDictionary for keys() and values() methods
- Fixed matching of certificate name in ssl with Python2
- Attributes names and formatters are checked even if schema is not read by the server
- Fixed fractional time when parsing generalized time
- Specific decoder for Active Directory ObjectGuid and ObjectSid
- Added additional checking for unicode in Python 2
- Tested against Python 3.4.2, 2.7.8, 2.6.6
- Updated setuptools to 7.0

**# 0.9.5.4 - 2014.09.22**

- Fixed security issue in lazy connections (thanks Moritz)
- Added ldap3.utils.dn with parse_dn(dn) to verify dn compliance with RFC4514
- Added safe_dn(dn) to properly escape dn (if possible)
- Added ldap3.utils.uri with parse_uri(uri) to verify uri compliance with RFC4516
- Check for trailing slashes in hostname (thanks Dylan)
- Timeout for socket connect operation. Server.connect_timeout = seconds_to_wait_for_establishing_connection (thanks Florian)
- Closing socket error doesn't raise exception anymore
- ServerPool can be implicity defined with a list of server names (even when defining a connection)

**# 0.9.5.3 - 2014.08.24**

- elements returned in schema and dsa info are in a case insensitive dictionary (can be changed in ldap3.CASE_INSENSITIVE_SCHEMA_NAMES = True|False)
- attributes name returned in searches are now case insensitive (can be changed in ldap3.CASE_INSENSITIVE_ATTRIBUTE_NAMES = True|False)
- change parameter name from separe_rdn to separate_rdn in ldap3.utils.conv.to_dn()

- sync dev from Bitbucket to GitHub
- schema attributes are explicitly read (useful for Active directory and 389 Directory Server)
- new extended operation: list_replicas (Novell)
- new extended operation: get_replica_info (Novell)
- new extended operation: partition_entry_count (Novell)
- renamed convert_to_ldif() to _convert_to_ldif()

**# 0.9.5.2 - 2014.08.05**

- fixed LDAPOperationResult.__str__ (thanks David)
- added to_dn() in utils.conv to convert a dn string to a list of components (strings or tuples)
- added __version__ in ldap3
- don't raise exception if the schema cannot be read in unauthenticated state
- server.address_info is now a property

**# 0.9.5.1 - 2014.08.02**

- getaddrinfo called only once
- real_server machinery removed - messageId is now global and monotonic for the whole library
- attributes are returned formatted if schema is read and check_names = True, removed checked_attributes
- bind result is populated again when successful (was removed in 0.9.2.1)
- exception is now raised if you receive multiple extended response to a single extended request. This is not allowed by RFC 4511

**# 0.9.5 - 2014.07.22**

- added support for IPv6 (thanks Robert)
- auto_bind can be used even for establishing tls, possible values (defined in ldap3) are: AUTO_BIND_NONE, AUTO_BIND_NO_TLS, AUTO_BIND_TLS_AFTER_BIND, AUTO_BIND_TLS_BEFORE_BIND
- refactored extend package to use classes
- new extended operation: get_universal_password (Novell)
- new extended operation: set_universal_password (Novell)
- added parsing of hostname in scheme://hostname:hostport format. This has the precedence on the parameters (thanks Sorin)
- added extra checks when the schema is read (with the get_info parameter) but nothing is returned by the server
- updated setuptools to version 5.4.1
- when check_name is True and schema is read attributes are checked and formatted in "checked_attributes" as specified by RFCs following the server schema
- added formatter for generalizedTime syntax as specified in RFC4517 (ASN.1)
- custom formatter can be added in Server definition

**# 0.9.4.2 - 2014.07.03**

- Moved to Bitbucket + Mercurial

- Fixed import in core.tls package
- Removed unneeded imports

**# 0.9.4.1 - 2014.07.02**

- included missing extend package (thanks to debnet)

**# 0.9.4 - 2014.07.02**

- when running in python 3.4 or newer now Tls class uses SSLContext object with default secure setting
- added parameters ca_certs_path, ca_certs_data, local_private_key_password to Tls object creation, valid when using SSLContext
- in python 3.4 or newer the system CA certificates configuration can be used (just leave ca_cert_file, ca_certs_path and ca_certs_data set to None)
- removed TLSv1 as default for Tls connection
- upgraded backported ssl function from python 3.4.1 when using with python 2
- when creating a connection the server parameter can be a string: the name of the server to connect in cleartext on default port 389
- fixed bug in ldap3.util.conv.escape_bytes()
- attributes parameter in search can be a tuple
- check_names parameter in connection now defaults to True (so when schema info is available attribute and class name will be checked when performing LDAP operations)
- remove the connection.close() method - use connection.unbind()
- new exception LDAPExtensionError for signalling when the requestValue of extended operation is of an unknown ASN1 type
- exiting connection manager doesn't raise an exception if unbind is not successful (needed in long operations)
- new extended operation: modify_password (RFC3062)
- new extended operation: who_am_i (RFC4532)
- new extended operation: get_bind_dn (Novell)
- updated setuptools to version 5.3

**# 0.9.3.5 - 2014.06.22**

- Exception history in restartable strategy is printed when reached the maximum number of retries
- Fixed conditions on terminated_by_server unsolicited message
- Added python2.6 egg installation package

**# 0.9.3.4 - 2014.06.16**

- Exception can now be imported from ldap3 package
- Escape_bytes return '' for empty string instead of None (thanks Brian)
- Added exception history to restartable connection (not for infinite retries)
- Fixed start_tls retrying in restartable connection (thanks Brian)
- New exception LDAPMaximumRetriesError for signalling when the SyncRestartable Strategy has reached the maximum number of retries while performing an operation

- Inverted deleteoldrdn value in LDIF output (thanks Joseph)

**# 0.9.3.3 - 2014.06.01**

- Fixed a bug in LDIFProducer when using context manager for connection
- LDIF header in stream is added only whene there are actual data in the stream
- Now LDIF stream can be added to an existing file - version header will not be written if stream is not empty

**# 0.9.3.2 - 2014.05.30**

- Fixed a bug while reading schema
- Add an implicit open() when trying binding on a closed connection

**# 0.9.3.1 - 2014.05.28**

- Added stream capability to LDIFProducer strategy
- Customizable line separator for LDIF output
- Customizable sorting order for LDIF output
- object_class parameter is now optional in connection.add()
- Fixed objectClass attribute case sensitive dependency in add operation
- Added stream capability to response_to_ldif() while searching

**# 0.9.3 - 2014.05.20**

- Now the key in server.schema.attribute_type is the attribute name (was the oid)
- Now the key in server.schema.object_classes is the class name (was the oid)
- Added check_names to Connection definition to have name of attributes and of object class checked against the schema
- Updated setuptools to 3.6
- Added wheel installation format
- Added raise_exceptions mode for connection
- Exception hierarchy reworked
- Added locking to Server object (for multithreading)

**# 0.9.2.2 - 2014.04.30**

- fixed a bug from 0.9.1 that broke start_tls() (thanks Mark)

**# 0.9.2.1 - 2014.04.28**

- fixed a bug in 0.9.2 that allowed only string attributes in add, modify and compare operations (thank Mladen)

**# 0.9.2 - 2014.04.26**

- changed return value in get_response from response to (response, result) - helpful for multi-threaded connections
- added ReusableStrategy for pooling connections
- refined docstrings (thanks Will)
- result and response attributes don't overlap anymore. Operation result is only in result attribute.

- fixed search for binary values (thanks Marcin)
- added convenience function to convert bytes to LDAP binary value string format for search filter

**# 0.9.1 - 2014.03.30**

- added laziness flag to test suite
- changed ServerPool signature to accept active and exhaust parameters
- removed unneeded start_listen parameter
- added 'lazy' parameter to open, to bind and to unbind a connection only when an effective operation is performed
- fixed start_tls in SyncWaitRestartable strategy
- fixed certificate name checking while opening an ssl connection
- fixed syntax error during installation
- socket operations now raises proper exception, not generic LDAPException (thanks Joseph)
- tested against Python 3.4, 3.3, 2.7, 2.6
- updated setuptools to 3.3

**# 0.9.0 - 2014.03.20**

- PEP8 compliance
- added ldap3.compat package with older (non PEP8 compliant) signatures
- renamed ldap3.abstraction to ldap3.abstract
- moved connection.py, server.py and tls.py files to ldap3.core
- fixed SyncWaitRestartableStrategy (thanks Christoph)

**# 0.8.3 - 2014.03.08**

- added SyncWaitRestartable strategy
- removed useless forceBind parameter
- usage statistics updated with restartable success/failure counters and open/closed/wrapped socket counters

**# 0.8.2 - 2014.03.04**

- Added refresh() method to Entry object to read again the attributes from the Reader in the abstraction layer
- Fixed Python 2.6 issues
- Fixed test suite for Python 2.6

**# 0.8.1 - 2014.02.12**

- Changed exceptions returned by the library to LDAPException, a subclass of Exception.
- Fixed documentation typos

**# 0.8.0 - 2014.02.08**

- Added abstraction layer (for searching, read only)
- Added context manager to Connection class
- Added readOnly parameter to Connection class
- Fixed a bug in search with 'less than' parameter

- Remove validation of available SSL protocols because different Python interpreters can use different ssl packages

**# 0.7.3 - 2014.01.05**

- Added SASL DIGEST-MD5 support
- Moved to intrapackage (relative) imports

**# 0.7.2 - 2013.12.30**

- Fixed a bug when parentheses are used in search filter as ASCII escaped sequences

**# 0.7.1 - 2013.12.21**

- Completed support for LDIF as per RFC2849
- Added new LDIF_PRODUCER strategy to generate LDIF-CHANGE stream
- Fixed a bug in the autoReferral feature when controls where used in operation

**# 0.7.0 - 2013.12.12**

- Added support for LDIF as per RFC2849
- Added LDIF-CONTENT compliant search responses
- Added exception when using autoBind if connection is not successful

**# 0.6.7 - 2013.12.03**

- Fixed exception when DSA is not willing to return rootDSE and schema info

**# 0.6.6 - 2013.11.13**

- Added parameters to test suite

**# 0.6.5 - 2013.11.05**

- Modified rawAttributes decoding, now null (empty) values are returned

**# 0.6.4 - 2013.10.16**

- Added simple paged search as per RFC2696
- Controls return values are decoded and stored in result attribute of connection

**# 0.6.3 - 2013.10.07**

- Added Extesible Filter syntax to search filter
- Fixed exception while closing connection in AsyncThreaded strategy

**# 0.6.2 - 2013.10.01**

- Fix for referrals in searchRefResult
- Disabled schema reading on Active Directory

**# 0.6.1 - 2013.09.22**

- Experimental support for Python 2 - no unicode
- Added backport of ssl.match_name for Python 2
- Minor fixes for using the client in Python 2
- Fix for getting schema info with AsyncThreaded strategy

**# 0.6.0 - 2013.09.16**

- Moved to beta!
- Added support site hosted on www.assembla.com
- Added public svn repository on www.assembla.com
- Added getInfo to server object, parameter can be: GET_NO_INFO, GET_DSA_INFO, GET_SCHEMA_INFO, GET_ALL_INFO
- Added method to read the schema from the server. Schema is decoded and returned in different dictionaries of the server.schema object
- Updated connection usage info (elapsed time is now computed when connection is closed)
- Updated OID dictionary with extensions and controls from Active Directory specifications.

**# 0.5.3 - 2013.09.03**

- Added getOperationalAttributes boolean to Search operation to fetch the operational attributes during search
- Added increment operation to modify operation as per RFC4525
- Added dictionary of OID descriptions (for DSE and schema decoding)
- Added method to get Info from DSE (returned in server.info object)
- Modified exceptions for sending controls in LDAP request
- Added connection usage (in connection.usage if collectUsage=True in connection definition)
- Fixed StartTls in asynchronous client strategy

**# 0.5.2 - 2013.08.27**

- Added SASLprep profile for validating password
- Fixed RFC4511 asn1 definitions

**# 0.5.1 - 2013.08.17**

- Refactored package structure
- Project description reformatted with reStructuredText
- Added Windows graphical installation

**# 0.5.0 - 2013.08.15**

- Added reference to LGPL v3 license
- Added Tls object to hold ssl/tls configuration
- Added StartTLS feature
- Added SASL feature
- Added SASL EXTERNAL mechanism
- Fixed Unbind
- connection.close is now an alias for connection.unbind

**# 0.4.4 - 2013.08.01**

- Added 'Controls' to all LDAP Requests
- Added Extended Request feature
- Added Intermediate Response feature

- Added namespace 'ldap3'

**# 0.4.3 - 2013.07.31**

- Test suite refactored
- Fixed single object search response error
- Changed attributes returned in search from tuple to dict
- Added 'raw_attributes' key in search response to hold undecoded (binary) attribute values read from ldap
- Added __repr__ for Server and Connection objects to re-create the object instance

**# 0.4.2 - 2013.07.29**

- Added autoReferral feature as per RFC4511 (4.1.10)
- Added allowedReferralHosts to conform to Security considerations of RFC4516

**# 0.4.1 - 2013.07.20**

- Add validation to Abandon operation
- Added connection.request to hold a dictionary of infos about last request
- Added info about outstanding operation in connection.strategy._oustanding
- Implemented RFC4515 for search filter coding and decoding
- Added a parser to build filter string from LdapMessage

**# 0.4.0 - 2013.07.15**

- Refactoring of the connection and strategy classes
- Added the ldap3.strategy namespace to contain client connection strategies
- Added ssl authentication
- Moved authentication parameters from Server object to Connection object
- Added ssl parameters to Server Object

**# 0.3.0 - 2013.07.14**

- Fixed AsyncThreaded strategy with _outstanding and _responses attributes to hold the pending requests and the not-yet-read responses
- Added Extended Operation
- Added "Unsolicited Notification" discover logic
- Added managing of "Notice of Disconnection" from server to properly close connection

**# 0.2.0 - 2013.07.13**

- Update setup with setuptools 0.7
- Docstrings added to class
- Removed ez_setup dependency
- Removed distribute dependency

**# 0.1.0 - 2013.07.12**

- Initial upload on pypi
- PyASN1 RFC4511 module completed and tested

- Synchronous client working properly
- Asynchronous client working but not fully tested
- Basic authentication working

# ldap3

## ldap3 package

### Subpackages

### ldap3.abstract package

### Submodules

### ldap3.abstract.attrDef module

**class** ldap3.abstract.attrDef.**AttrDef**(*name*, *key=None*, *validate=None*, *pre_query=None*, *post_query=None*, *default=NotImplemented*, *dereference_dn=None*, *description=None*, *mandatory=False*, *single_value=None*, *alias=None*)

Bases: object

Hold the definition of an attribute

**Parameters**

- **name** (*string*) – the real attribute name
- **key** (*string*) – the friendly name to use in queries and when accessing the attribute, default to the real attribute name
- **validate** (*callable*) – called to check if the value in the query is valid, the callable is called with the value parameter
- **pre_query** (*callable*) – called to transform values returned by search
- **post_query** (*callable*) – called to transform values returned by search
- **default** (*string, integer*) – value returned when the attribute is absent (defaults to NotImplemented to allow use of None as default)
- **dereference_dn** (*ObjectDef*) – reference to an ObjectDef instance. When the attribute value contains a dn it will be searched and substituted in the entry
- **description** (*string*) – custom attribute description
- **mandatory** (*boolean*) – specify if attribute is defined as mandatory in LDAP schema

### ldap3.abstract.attribute module

**class** ldap3.abstract.attribute.**Attribute**(*attr_def*, *entry*, *cursor*)

Bases: object

Attribute/values object, it includes the search result (after post_query transformation) of each attribute in an entry

Attribute object is read only

- values: contain the processed attribute values
- raw_values': contain the unprocessed attribute values

**value**

**Returns** The single value or a list of values of the attribute.

**class** ldap3.abstract.attribute.**OperationalAttribute**(*attr_def*, *entry*, *cursor*)
Bases: ldap3.abstract.attribute.Attribute

Operational attribute/values object. Include the search result of an operational attribute in an entry

OperationalAttribute object is read only

- values: contains the processed attribute values
- raw_values: contains the unprocessed attribute values

It may not have an AttrDef

**class** ldap3.abstract.attribute.**WritableAttribute**(*attr_def*, *entry*, *cursor*)
Bases: ldap3.abstract.attribute.Attribute

**add**(*values*)

**changes**

**delete**(*values*)

**discard**()

**remove**()

**set**(*values*)

**virtual**

### ldap3.abstract.cursor module

**class** ldap3.abstract.cursor.**Cursor**(*connection*, *object_def*, *get_operational_attributes=False*, *attributes=None*, *controls=None*)
Bases: object

**errors**

**failed**

**match**(*attributes*, *value*)
Return entries with text in one of the specified attributes

**match_dn**(*dn*)
Return entries with text in DN

**operations**

**remove**(*entry*)

**class** ldap3.abstract.cursor.**Operation**(*request*, *result*, *response*)
Bases: tuple

**request**
Alias for field number 0

**response**
Alias for field number 2

**result**
Alias for field number 1

**class** ldap3.abstract.cursor.**Reader**(*connection*, *object_def*, *base*, *query=''*, *components_in_and=True*, *sub_tree=True*, *get_operational_attributes=False*, *attributes=None*, *controls=None*)

Bases: ldap3.abstract.cursor.Cursor

Reader object to perform searches:

**Parameters**

- **connection** (*LDAPConnection*) – the LDAP connection object to use
- **object_def** (*ObjectDef*) – the ObjectDef of the LDAP object returned
- **query** (*str*) – the simplified query (will be transformed in an LDAP filter)
- **base** (*str*) – starting base of the search
- **components_in_and** (*bool*) – specify if assertions in the query must all be satisfied or not (AND/OR)
- **sub_tree** (*bool*) – specify if the search must be performed ad Single Level (False) or Whole SubTree (True)
- **get_operational_attributes** (*bool*) – specify if operational attributes are returned or not
- **controls** (*tuple*) – controls to be used in search

**attribute_class**
alias of Attribute

**clear**()
Clear the Reader search parameters

**components_in_and**

**entry_class**
alias of Entry

**entry_initial_status = 'Read'**

**query**

**reset**()
Clear all the Reader parameters

**search**(*attributes=None*)
Perform the LDAP search

**Returns** Entries found in search

**search_level**(*attributes=None*)
Perform the LDAP search operation with SINGLE_LEVEL scope

**Returns** Entries found in search

**search_object**(*entry_dn=None*, *attributes=None*)
Perform the LDAP search operation SINGLE_OBJECT scope

**Returns** Entry found in search

**search_paged**(*paged_size*, *paged_criticality=True*, *generator=True*, *attributes=None*)
Perform a paged search, can be called as an Iterator

**Parameters**

- **attributes** – optional attributes to search
- **paged_size** (*int*) – number of entries returned in each search
- **paged_criticality** (*bool*) – specify if server must not execute the search if it is not capable of paging searches
- **generator** (*bool*) – if True the paged searches are executed while generating the entries, if False all the paged searches are execute before returning the generator

**Returns** Entries found in search

**search_subtree**(*attributes=None*)
Perform the LDAP search operation WHOLE_SUBTREE scope

**Returns** Entries found in search

**class** ldap3.abstract.cursor.**Writer**(*connection*, *object_def*, *get_operational_attributes=False*, *attributes=None*, *controls=None*)
Bases: ldap3.abstract.cursor.Cursor

**attribute_class**
alias of WritableAttribute

**commit**(*refresh=True*)

**discard**()

**entry_class**
alias of WritableEntry

**entry_initial_status = 'Writable'**

**static from_cursor**(*cursor*, *connection=None*, *object_def=None*, *custom_validator=None*)

**static from_response**(*connection*, *object_def*, *response=None*)

**new**(*dn*)

**refresh_entry**(*entry*, *tries=4*, *seconds=2*)

### ldap3.abstract.entry module

**class** ldap3.abstract.entry.**Entry**(*dn*, *cursor*)
Bases: ldap3.abstract.entry.EntryBase

The Entry object contains a single LDAP entry. Attributes can be accessed either by sequence, by assignment or as dictionary keys. Keys are not case sensitive.

The Entry object is read only

•The DN is retrieved by _dn()

•The Reader reference is in _cursor()

•Raw attributes values are retrieved by the _ra_attributes and _raw_attribute() methods

**entry_writable**(*object_def=None*, *writer_cursor=None*, *attributes=None*, *custom_validator=None*)

**class** ldap3.abstract.entry.**EntryBase**(*dn*, *cursor*)
Bases: object

The Entry object contains a single LDAP entry. Attributes can be accessed either by sequence, by assignment or as dictionary keys. Keys are not case sensitive.

The Entry object is read only

- The DN is retrieved by _dn
- The cursor reference is in _cursor
- Raw attributes values are retrieved with _raw_attributes and the _raw_attribute() methods

**entry_attributes**

**entry_attributes_as_dict**

**entry_cursor**

**entry_definition**

**entry_dn**

**entry_mandatory_attributes**

**entry_raw_attribute**(*name*)

**Parameters** **name** – name of the attribute

**Returns** raw (unencoded) value of the attribute, None if attribute is not found

**entry_raw_attributes**

**entry_read_time**

**entry_status**

**entry_to_json**(*raw=False*, *indent=4*, *sort=True*, *stream=None*, *checked_attributes=True*, *include_empty=True*)

**entry_to_ldif**(*all_base64=False*, *line_separator=None*, *sort_order=None*, *stream=None*)

**class** ldap3.abstract.entry.**EntryState**(*dn*, *cursor*)
Bases: object

Contains data on the status of the entry. Does not pollute the Entry __dict__.

**set_status**(*status*)

**class** ldap3.abstract.entry.**WritableEntry**(*dn*, *cursor*)
Bases: ldap3.abstract.entry.EntryBase

**entry_changes**

**entry_commit_changes**(*refresh=True*, *controls=None*, *clear_history=True*)

**entry_delete**()

**entry_discard_changes**()

**entry_move**(*destination_dn*)

**entry_refresh**(*tries=4*, *seconds=2*)
Refreshes the entry from the LDAP Server

**entry_rename**(*new_name*)

**entry_virtual_attributes**

### ldap3.abstract.objectDef module

**class** ldap3.abstract.objectDef.**ObjectDef**(*object_class=None*, *schema=None*, *custom_validator=None*)

Bases: object

Represent an object in the LDAP server. AttrDefs are stored in a dictionary; the key is the friendly name defined in AttrDef.

AttrDefs can be added and removed using the += ad -= operators

ObjectDef can be accessed either as a sequence and a dictionary. When accessed the whole AttrDef instance is returned

**add_attribute**(*definition=None*)

Add an AttrDef to the ObjectDef. Can be called with the += operator. :param definition: the AttrDef object to add, can also be a string containing the name of attribute to add. Can be a list of both

**add_from_schema**(*attribute_name*, *mandatory=False*)

**clear_attributes**()

Empty the ObjectDef attribute list

**remove_attribute**(*item*)

Remove an AttrDef from the ObjectDef. Can be called with the -= operator. :param item: the AttrDef to remove, can also be a string containing the name of attribute to remove

### Module contents

### ldap3.core package

### Submodules

### ldap3.core.connection module

**class** ldap3.core.connection.**Connection**(*server*, *user=None*, *password=None*, *auto_bind='NONE'*, *version=3*, *authentication=None*, *client_strategy='SYNC'*, *auto_referrals=True*, *auto_range=True*, *sasl_mechanism=None*, *sasl_credentials=None*, *check_names=True*, *collect_usage=False*, *read_only=False*, *lazy=False*, *raise_exceptions=False*, *pool_name=None*, *pool_size=None*, *pool_lifetime=None*, *fast_decoder=True*, *receive_timeout=None*, *return_empty_attributes=True*, *use_referral_cache=False*, *auto_escape=True*, *auto_encode=True*, *pool_keepalive=None*)

Bases: object

Main ldap connection class.

Controls, if used, must be a list of tuples. Each tuple must have 3 elements, the control OID, a boolean meaning if the control is critical, a value.

If the boolean is set to True the server must honor the control or refuse the operation

Mixing controls must be defined in controls specification (as per RFC 4511)

**Parameters**

- **server** (*Server, str*) – the Server object to connect to
- **user** (*str*) – the user name for simple authentication
- **password** (*str*) – the password for simple authentication
- **auto_bind** (*int, can be one of AUTO_BIND_NONE, AUTO_BIND_NO_TLS, AUTO_BIND_TLS_BEFORE_BIND, AUTO_BIND_TLS_AFTER_BIND as specified in ldap3*) – specify if the bind will be performed automatically when defining the Connection object
- **version** (*int*) – LDAP version, default to 3
- **authentication** (*int, can be one of AUTH_ANONYMOUS, AUTH_SIMPLE or AUTH_SASL, as specified in ldap3*) – type of authentication
- **client_strategy** (*can be one of STRATEGY_SYNC, STRATEGY_ASYNC_THREADED, STRATEGY_LDIF_PRODUCER, STRATEGY_SYNC_RESTARTABLE, STRATEGY_REUSABLE_THREADED as specified in ldap3*) – communication strategy used in the Connection
- **auto_referrals** (*bool*) – specify if the connection object must automatically follow referrals
- **sasl_mechanism** (*str*) – mechanism for SASL authentication, can be one of 'EXTERNAL', 'DIGEST-MD5', 'GSSAPI', 'PLAIN'
- **sasl_credentials** (*tuple*) – credentials for SASL mechanism
- **check_names** (*bool*) – if True the library will check names of attributes and object classes against the schema. Also values found in entries will be formatted as indicated by the schema
- **collect_usage** (*bool*) – collect usage metrics in the usage attribute
- **read_only** (*bool*) – disable operations that modify data in the LDAP server
- **lazy** (*bool*) – open and bind the connection only when an actual operation is performed
- **raise_exceptions** (*bool*) – raise exceptions when operations are not successful, if False operations return False if not successful but not raise exceptions
- **pool_name** (*str*) – pool name for pooled strategies
- **pool_size** (*int*) – pool size for pooled strategies
- **pool_lifetime** (*int*) – pool lifetime for pooled strategies
- **use_referral_cache** (*bool*) – keep referral connections open and reuse them
- **auto_escape** – automatic escaping of filter values
- **auto_encode** – automatic encoding of attribute values

**abandon** (*message_id*, *controls=None*)
Abandon the operation indicated by message_id

**add** (*dn*, *object_class=None*, *attributes=None*, *controls=None*)
Add dn to the DIT, object_class is None, a class name or a list of class names.

Attributes is a dictionary in the form 'attr': 'val' or 'attr': ['val1', 'val2', ...] for multivalued attributes

**bind**(*read_server_info=True*, *controls=None*)
Bind to ldap Server with the authentication method and the user defined in the connection

**Parameters**

- **read_server_info** – reads info from server
- **controls** (*list of tuple*) – LDAP controls to send along with the bind operation

**Returns** bool

**compare**(*dn*, *attribute*, *value*, *controls=None*)
Perform a compare operation

**delete**(*dn*, *controls=None*)
Delete the entry identified by the DN from the DIB.

**do_ntlm_bind**(*controls*)

**do_sasl_bind**(*controls*)

**entries**

**extended**(*request_name*, *request_value=None*, *controls=None*, *no_encode=None*)
Performs an extended operation

**modify**(*dn*, *changes*, *controls=None*)
Modify attributes of entry

•changes is a dictionary in the form {'attribute1': change), 'attribute2': [change, change, ...], ...}

•change is (operation, [value1, value2, ...])

•operation is 0 (MODIFY_ADD), 1 (MODIFY_DELETE), 2 (MODIFY_REPLACE), 3 (MODIFY_INCREMENT)

**modify_dn**(*dn*, *relative_dn*, *delete_old_dn=True*, *new_superior=None*, *controls=None*)
Modify DN of the entry or performs a move of the entry in the DIT.

**rebind**(*user=None*, *password=None*, *authentication=None*, *sasl_mechanism=None*, *sasl_credentials=None*, *read_server_info=True*, *controls=None*)

**refresh_server_info**()

**repr_with_sensitive_data_stripped**()

**response_to_file**(*target*, *raw=False*, *indent=4*, *sort=True*)

**response_to_json**(*raw=False*, *search_result=None*, *indent=4*, *sort=True*, *stream=None*, *checked_attributes=True*, *include_empty=True*)

**response_to_ldif**(*search_result=None*, *all_base64=False*, *line_separator=None*, *sort_order=None*, *stream=None*)

**search**(*search_base*, *search_filter*, *search_scope='SUBTREE'*, *dereference_aliases='ALWAYS'*, *attributes=None*, *size_limit=0*, *time_limit=0*, *types_only=False*, *get_operational_attributes=False*, *controls=None*, *paged_size=None*, *paged_criticality=False*, *paged_cookie=None*, *auto_escape=None*)
Perform an ldap search:

•If attributes is empty no attribute is returned

•If attributes is ALL_ATTRIBUTES all attributes are returned

•If paged_size is an int greater than 0 a simple paged search is tried as described in RFC2696 with the specified size

•If paged is 0 and cookie is present the search is abandoned on server

- Cookie is an opaque string received in the last paged search and must be used on the next paged search response
- If lazy == True open and bind will be deferred until another LDAP operation is performed
- If mssing_attributes == True then an attribute not returned by the server is set to None
- If auto_escape is set it overrides the Connection auto_escape

**start_tls**(*read_server_info=True*)

**stream**
Used by the LDIFProducer strategy to accumulate the ldif-change operations with a single LDIF header :return: reference to the response stream if defined in the strategy.

**unbind**(*controls=None*)
Unbind the connected user. Unbind implies closing session as per RFC4511 (4.3)

**Parameters** **controls** – LDAP controls to send along with the bind operation

**usage**
Usage statistics for the connection. :return: Usage object

### ldap3.core.exceptions module

**exception** ldap3.core.exceptions.**LDAPAdminLimitExceededResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAffectMultipleDSASResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAliasDereferencingProblemResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAliasProblemResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAssertionFailedResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAttributeError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError, TypeError

**exception** ldap3.core.exceptions.**LDAPAttributeOrValueExistsResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAuthMethodNotSupportedResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPAuthorizationDeniedResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPBindError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPBusyResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPCanceledResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPCannotCancelResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPCertificateError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPChangeError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPCommunicationError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPConfidentialityRequiredResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPConfigurationError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPConfigurationParameterError**
Bases: ldap3.core.exceptions.LDAPConfigurationError

**exception** ldap3.core.exceptions.**LDAPConnectionIsReadOnlyError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPConnectionPoolNameIsMandatoryError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPConnectionPoolNotStartedError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPConstraintViolationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPControlError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPCursorError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPDefinitionError**
Bases: ldap3.core.exceptions.LDAPConfigurationError

**exception** ldap3.core.exceptions.**LDAPESyncRefreshRequiredResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPEntryAlreadyExistsResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPException**
Bases: Exception

**exception** ldap3.core.exceptions.**LDAPExceptionError**
Bases: ldap3.core.exceptions.LDAPException

**exception** ldap3.core.exceptions.**LDAPExtensionError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPInappropriateAuthenticationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPInappropriateMatchingResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPInsufficientAccessRightsResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPInvalidAttributeSyntaxResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPInvalidCredentialsResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPInvalidDNSyntaxResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPInvalidDereferenceAliasesError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPInvalidDnError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPInvalidFilterError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPInvalidHashAlgorithmError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPInvalidPortError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPInvalidScopeError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPInvalidServerError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPInvalidTlsSpecificationError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPInvalidValueError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPKeyError**
Bases: ldap3.core.exceptions.LDAPExceptionError, KeyError, AttributeError

**exception** ldap3.core.exceptions.**LDAPLCUPInvalidDataResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPLCUPReloadRequiredResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPLCUPResourcesExhaustedResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPLCUPSecurityViolationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPLCUPUnsupportedSchemeResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPLDIFError**

Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPLoopDetectedResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPMaximumRetriesError**

Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPMetricsError**

Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPNamingViolationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPNoSuchAttributeResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPNoSuchObjectResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPNoSuchOperationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPNotAllowedOnNotLeafResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)

Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPNotAllowedOnRDNResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPObjectClassError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPObjectClassModsProhibitedResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPObjectClassViolationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPObjectError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ValueError

**exception** ldap3.core.exceptions.**LDAPOperationResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPException

**exception** ldap3.core.exceptions.**LDAPOperationsErrorResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPOtherResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPPackageUnavailableError**
Bases: ldap3.core.exceptions.LDAPConfigurationError, ImportError

**exception** ldap3.core.exceptions.**LDAPPasswordIsMandatoryError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPProtocolErrorResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPReferralError**
Bases: ldap3.core.exceptions.LDAPCommunicationError

**exception** ldap3.core.exceptions.**LDAPReferralResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPResponseTimeoutError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPSASLBindInProgressError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPSASLBindInProgressResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPSASLMechanismNotSupportedError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPSASLPrepError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPSSLConfigurationError**
Bases: ldap3.core.exceptions.LDAPConfigurationError

**exception** ldap3.core.exceptions.**LDAPSSLNotSupportedError**
Bases: ldap3.core.exceptions.LDAPExceptionError, ImportError

**exception** ldap3.core.exceptions.**LDAPSchemaError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPServerPoolError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPServerPoolExhaustedError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPSessionTerminatedByServerError**
Bases: ldap3.core.exceptions.LDAPCommunicationError

**exception** ldap3.core.exceptions.**LDAPSizeLimitExceededResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPSocketCloseError**
Bases: ldap3.core.exceptions.LDAPCommunicationError

**exception** ldap3.core.exceptions.**LDAPSocketOpenError**
Bases: ldap3.core.exceptions.LDAPCommunicationError

**exception** ldap3.core.exceptions.**LDAPSocketReceiveError**
Bases: ldap3.core.exceptions.LDAPCommunicationError, OSError

**exception** ldap3.core.exceptions.**LDAPSocketSendError**
Bases: ldap3.core.exceptions.LDAPCommunicationError, OSError

**exception** ldap3.core.exceptions.**LDAPStartTLSError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPStrongerAuthRequiredResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPTimeLimitExceededResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPTooLateResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPTransactionError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPUnavailableCriticalExtensionResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPUnavailableResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPUndefinedAttributeTypeResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPUnknownAuthenticationMethodError**
Bases: ldap3.core.exceptions.LDAPConfigurationError

**exception** ldap3.core.exceptions.**LDAPUnknownRequestError**
Bases: ldap3.core.exceptions.LDAPCommunicationError

**exception** ldap3.core.exceptions.**LDAPUnknownResponseError**
Bases: ldap3.core.exceptions.LDAPCommunicationError

**exception** ldap3.core.exceptions.**LDAPUnknownStrategyError**
Bases: ldap3.core.exceptions.LDAPConfigurationError

**exception** ldap3.core.exceptions.**LDAPUnwillingToPerformResult**(*result=None*, *description=None*, *dn=None*, *message=None*, *response_type=None*, *response=None*)
Bases: ldap3.core.exceptions.LDAPOperationResult

**exception** ldap3.core.exceptions.**LDAPUserNameIsMandatoryError**
Bases: ldap3.core.exceptions.LDAPExceptionError

**exception** ldap3.core.exceptions.**LDAPUserNameNotAllowedError**
Bases: ldap3.core.exceptions.LDAPExceptionError

ldap3.core.exceptions.**communication_exception_factory**(*exc_to_raise*, *exc*)
Generates a new exception class of the requested type (subclass of LDAPCommunication) merged with the exception raised by the interpreter

ldap3.core.exceptions.**start_tls_exception_factory**(*exc_to_raise*, *exc*)
Generates a new exception class of the requested type (subclass of LDAPCommunication) merged with the exception raised by the interpreter

### ldap3.core.pooling module

**class** ldap3.core.pooling.**ServerPool**(*servers=None*, *pool_strategy='ROUND_ROBIN'*, *active=True*, *exhaust=False*)
Bases: object

**add**(*servers*)

**get_current_server**(*connection*)

**get_server**(*connection*)

**initialize**(*connection*)

**remove**(*server*)

**class** ldap3.core.pooling.**ServerPoolState**(*server_pool*)
Bases: object

**find_active_random_server**()

**find_active_server**(*starting*)

**get_current_server**()

**get_server**()

**refresh**()

### ldap3.core.results module

### ldap3.core.server module

**class** ldap3.core.server.**Server**(*host*, *port=None*, *use_ssl=False*, *allowed_referral_hosts=None*, *get_info='SCHEMA'*, *tls=None*, *formatter=None*, *connect_timeout=None*, *mode='IP_V6_PREFERRED'*, *validator=None*)

Bases: object

LDAP Server definition class

Allowed_referral_hosts can be None (default), or a list of tuples of allowed servers ip address or names to contact while redirecting search to referrals.

The second element of the tuple is a boolean to indicate if authentication to that server is allowed; if False only anonymous bind will be used.

Per RFC 4516. Use [('*', False)] to allow any host with anonymous bind, use [('*', True)] to allow any host with same authentication of Server.

**address_info**

**attach_dsa_info**(*dsa_info=None*)

**attach_schema_info**(*dsa_schema=None*)

**candidate_addresses**()

**check_availability**()
Tries to open, connect and close a socket to specified address and port to check availability. Timeout in seconds is specified in CHECK_AVAILABITY_TIMEOUT if not specified in the Server object

**static from_definition**(*host*, *dsa_info*, *dsa_schema*, *port=None*, *use_ssl=False*, *formatter=None*, *validator=None*)
Define a dummy server with preloaded schema and info :param host: host name :param dsa_info: DsaInfo preloaded object or a json formatted string or a file name :param dsa_schema: SchemaInfo preloaded object or a json formatted string or a file name :param port: dummy port :param use_ssl: use_ssl :param formatter: custom formatter :return: Server object

**get_info_from_server**(*connection*)
reads info from DSE and from subschema

**info**

**static next_message_id**()
LDAP messageId is unique for all connections to same server

**reset_availability**()

**schema**

**update_availability**(*address*, *available*)

### ldap3.core.timezone module

**class** ldap3.core.timezone.**OffsetTzInfo**(*offset*, *name*)

Bases: datetime.tzinfo

Fixed offset in minutes east from UTC

**dst**(*dt*)

**tzname**(*dt*)

**utcoffset**(*dt*)

### ldap3.core.tls module

**class** ldap3.core.tls.**Tls**(*local_private_key_file=None*, *local_certificate_file=None*, *validate=0*, *version=None*, *ca_certs_file=None*, *valid_names=None*, *ca_certs_path=None*, *ca_certs_data=None*, *local_private_key_password=None*, *ciphers=None*, *sni=None*)

Bases: object

tls/ssl configuration for Server object Starting from python 2.7.9 and python 3.4 uses the SSLContext object that tries to read the CAs defined at system level ca_certs_path and ca_certs_data are valid only when using SSLContext local_private_key_password is valid only when using SSLContext sni is the server name for Server Name Indication (when available)

**start_tls**(*connection*)

**wrap_socket**(*connection*, *do_handshake=False*)

Adds TLS to the connection socket

ldap3.core.tls.**check_hostname**(*sock*, *server_name*, *additional_names*)

### ldap3.core.usage module

**class** ldap3.core.usage.**ConnectionUsage**

Bases: object

Collect statistics on connection usage

**elapsed_time**

**reset**()

**start**(*reset=True*)

**stop**()

**update_received_message**(*length*)

**update_transmitted_message**(*message*, *length*)

### Module contents

### ldap3.extend package

### Subpackages

### ldap3.extend.microsoft package

### Submodules

### ldap3.extend.microsoft.addMembersToGroups module

ldap3.extend.microsoft.addMembersToGroups.**ad_add_members_to_groups**(*connection*, *members_dn*, *groups_dn*, *fix=True*)

**Parameters**

- **connection** – a bound Connection object
- **members_dn** – the list of members to add to groups
- **groups_dn** – the list of groups where members are to be added
- **fix** – checks for group existence and already assigned members

**Returns** a boolean where True means that the operation was successful and False means an error has happened

Establishes users-groups relations following the Active Directory rules: users are added to the member attribute of groups. Raises LDAPInvalidDnError if members or groups are not found in the DIT.

### ldap3.extend.microsoft.dirSync module

**class** ldap3.extend.microsoft.dirSync.**DirSync**(*connection*, *sync_base*, *sync_filter*, *attributes*, *cookie*, *object_security*, *ancestors_first*, *public_data_only*, *incremental_values*, *max_length*, *hex_guid*)

Bases: object

**loop**()

### ldap3.extend.microsoft.modifyPassword module

ldap3.extend.microsoft.modifyPassword.**ad_modify_password**(*connection*, *user_dn*, *new_password*, *old_password*, *controls=None*)

### ldap3.extend.microsoft.removeMembersFromGroups module

ldap3.extend.microsoft.removeMembersFromGroups.**ad_remove_members_from_groups**(*connection*, *members_dn*, *groups_dn*, *fix*)

**Parameters**

- **connection** – a bound Connection object
- **members_dn** – the list of members to remove from groups
- **groups_dn** – the list of groups where members are to be removed
- **fix** – checks for group existence and existing members

**Returns** a boolean where True means that the operation was successful and False means an error has happened

Removes users-groups relations following the Activwe Directory rules: users are removed from groups' member attribute

### ldap3.extend.microsoft.unlockAccount module

ldap3.extend.microsoft.unlockAccount.**ad_unlock_account**(*connection*, *user_dn*, *controls=None*)

### Module contents

### ldap3.extend.novell package

### Submodules

### ldap3.extend.novell.addMembersToGroups module

ldap3.extend.novell.addMembersToGroups.**edir_add_members_to_groups**(*connection*, *members_dn*, *groups_dn*, *fix*, *transaction*)

**Parameters**

- **connection** – a bound Connection object
- **members_dn** – the list of members to add to groups
- **groups_dn** – the list of groups where members are to be added
- **fix** – checks for inconsistences in the users-groups relation and fixes them
- **transaction** – activates an LDAP transaction

**Returns** a boolean where True means that the operation was successful and False means an error has happened

Establishes users-groups relations following the eDirectory rules: groups are added to securityEquals and groupMembership attributes in the member object while members are added to member and equivalentToMe attributes in the group object. Raises LDAPInvalidDnError if members or groups are not found in the DIT.

### ldap3.extend.novell.checkGroupsMemberships module

ldap3.extend.novell.checkGroupsMemberships.**edir_check_groups_memberships**(*connection*, *members_dn*, *groups_dn*, *fix*, *transaction*)

**Parameters**

- **connection** – a bound Connection object
- **members_dn** – the list of members to check
- **groups_dn** – the list of groups to check
- **fix** – checks for inconsistences in the users-groups relation and fixes them
- **transaction** – activates an LDAP transaction when fixing

**Returns** a boolean where True means that the operation was successful and False means an error has happened

Checks and fixes users-groups relations following the eDirectory rules: groups are checked against 'groupMembership' attribute in the member object while members are checked against 'member' attribute in the group object. Raises LDAPInvalidDnError if members or groups are not found in the DIT.

### ldap3.extend.novell.endTransaction module

**class** ldap3.extend.novell.endTransaction.**EndTransaction**(*connection*, *commit=True*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

**set_response**()

### ldap3.extend.novell.getBindDn module

**class** ldap3.extend.novell.getBindDn.**GetBindDn**(*connection*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.novell.listReplicas module

**class** ldap3.extend.novell.listReplicas.**ListReplicas**(*connection*, *server_dn*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.novell.nmasGetUniversalPassword module

**class** ldap3.extend.novell.nmasGetUniversalPassword.**NmasGetUniversalPassword**(*connection*, *user*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.novell.nmasSetUniversalPassword module

**class** ldap3.extend.novell.nmasSetUniversalPassword.**NmasSetUniversalPassword**(*connection*, *user*, *new_password*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.novell.partition_entry_count module

**class** ldap3.extend.novell.partition_entry_count.**PartitionEntryCount**(*connection*, *partition_dn*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.novell.removeMembersFromGroups module

ldap3.extend.novell.removeMembersFromGroups.**edir_remove_members_from_groups**(*connection*, *members_dn*, *groups_dn*, *fix*, *transaction*)

**Parameters**

- **connection** – a bound Connection object
- **members_dn** – the list of members to remove from groups
- **groups_dn** – the list of groups where members are to be removed
- **fix** – checks for inconsistences in the users-groups relation and fixes them
- **transaction** – activates an LDAP transaction

**Returns** a boolean where True means that the operation was successful and False means an error has happened

Removes users-groups relations following the eDirectory rules: groups are removed from securityEquals and groupMembership attributes in the member object while members are removed from member and equivalentToMe attributes in the group object. Raises LDAPInvalidDnError if members or groups are not found in the DIT.

### ldap3.extend.novell.replicaInfo module

**class** ldap3.extend.novell.replicaInfo.**ReplicaInfo**(*connection*, *server_dn*, *partition_dn*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.novell.startTransaction module

**class** ldap3.extend.novell.startTransaction.**StartTransaction**(*connection*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

**set_response**()

### Module contents

### ldap3.extend.standard package

### Submodules

### ldap3.extend.standard.PagedSearch module

ldap3.extend.standard.PagedSearch.**paged_search_accumulator**(*connection*, *search_base*, *search_filter*, *search_scope='SUBTREE'*, *dereference_aliases='ALWAYS'*, *attributes=None*, *size_limit=0*, *time_limit=0*, *types_only=False*, *get_operational_attributes=False*, *controls=None*, *paged_size=100*, *paged_criticality=False*)

ldap3.extend.standard.PagedSearch.**paged_search_generator**(*connection*, *search_base*, *search_filter*, *search_scope='SUBTREE'*, *dereference_aliases='ALWAYS'*, *attributes=None*, *size_limit=0*, *time_limit=0*, *types_only=False*, *get_operational_attributes=False*, *controls=None*, *paged_size=100*, *paged_criticality=False*)

### ldap3.extend.standard.PersistentSearch module

**class** ldap3.extend.standard.PersistentSearch.**PersistentSearch**(*connection*, *search_base*, *search_filter*, *search_scope*, *dereference_aliases*, *attributes*, *size_limit*, *time_limit*, *controls*, *changes_only*, *events_type*, *notifications*, *streaming*, *callback*)

Bases: object

**next**()

**start**()

**stop**()

### ldap3.extend.standard.modifyPassword module

**class** ldap3.extend.standard.modifyPassword.**ModifyPassword**(*connection*, *user=None*, *old_password=None*, *new_password=None*, *hash_algorithm=None*, *salt=None*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### ldap3.extend.standard.whoAmI module

**class** ldap3.extend.standard.whoAmI.**WhoAmI**(*connection*, *controls=None*)

Bases: ldap3.extend.operation.ExtendedOperation

**config**()

**populate_result**()

### Module contents

### Submodules

### ldap3.extend.operation module

**class** ldap3.extend.operation.**ExtendedOperation**(*connection*, *controls=None*)
Bases: object

**config**()

**decode_response**()

**populate_result**()

**send**()

**set_response**()

### Module contents

**class** ldap3.extend.**ExtendedOperationContainer**(*connection*)
Bases: object

**class** ldap3.extend.**ExtendedOperationsRoot**(*connection*)
Bases: ldap3.extend.ExtendedOperationContainer

**class** ldap3.extend.**MicrosoftExtendedOperations**(*connection*)
Bases: ldap3.extend.ExtendedOperationContainer

**add_members_to_groups**(*members*, *groups*, *fix=True*)

**dir_sync**(*sync_base*, *sync_filter='(objectclass=*)'*, *attributes='*'*, *cookie=None*, *object_security=False*, *ancestors_first=True*, *public_data_only=False*, *incremental_values=True*, *max_length=2147483647*, *hex_guid=False*)

**modify_password**(*user*, *new_password*, *old_password=None*, *controls=None*)

**remove_members_from_groups**(*members*, *groups*, *fix=True*)

**unlock_account**(*user*)

**class** ldap3.extend.**NovellExtendedOperations**(*connection*)
Bases: ldap3.extend.ExtendedOperationContainer

**add_members_to_groups**(*members*, *groups*, *fix=True*, *transaction=True*)

**check_groups_memberships**(*members*, *groups*, *fix=False*, *transaction=True*)

**end_transaction**(*commit=True*, *controls=None*)

**get_bind_dn**(*controls=None*)

**get_universal_password**(*user*, *controls=None*)

**list_replicas**(*server_dn*, *controls=None*)

**partition_entry_count**(*partition_dn*, *controls=None*)

**remove_members_from_groups**(*members*, *groups*, *fix=True*, *transaction=True*)

**replica_info**(*server_dn*, *partition_dn*, *controls=None*)

**set_universal_password**(*user*, *new_password=None*, *controls=None*)

**start_transaction**(*controls=None*)

**class** ldap3.extend.**StandardExtendedOperations**(*connection*)
Bases: ldap3.extend.ExtendedOperationContainer

**modify_password**(*user=None*, *old_password=None*, *new_password=None*, *hash_algorithm=None*, *salt=None*, *controls=None*)

**paged_search**(*search_base*, *search_filter*, *search_scope='SUBTREE'*, *dereference_aliases='ALWAYS'*, *attributes=None*, *size_limit=0*, *time_limit=0*, *types_only=False*, *get_operational_attributes=False*, *controls=None*, *paged_size=100*, *paged_criticality=False*, *generator=True*)

**persistent_search**(*search_base=''*, *search_filter='(objectclass=*)'*, *search_scope='SUBTREE'*, *dereference_aliases='NEVER'*, *attributes='*'*, *size_limit=0*, *time_limit=0*, *controls=None*, *changes_only=True*, *show_additions=True*, *show_deletions=True*, *show_modifications=True*, *show_dn_modifications=True*, *notifications=True*, *streaming=True*, *callback=None*)

**who_am_i**(*controls=None*)

### ldap3.operation package

#### Submodules

#### ldap3.operation.abandon module

ldap3.operation.abandon.**abandon_operation**(*msg_id*)

ldap3.operation.abandon.**abandon_request_to_dict**(*request*)

#### ldap3.operation.add module

ldap3.operation.add.**add_operation**(*dn*, *attributes*, *auto_encode*, *schema=None*, *validator=None*)

ldap3.operation.add.**add_request_to_dict**(*request*)

ldap3.operation.add.**add_response_to_dict**(*response*)

#### ldap3.operation.bind module

ldap3.operation.bind.**bind_operation**(*version*, *authentication*, *name=''*, *password=None*, *sasl_mechanism=None*, *sasl_credentials=None*)

ldap3.operation.bind.**bind_request_to_dict**(*request*)

ldap3.operation.bind.**bind_response_operation**(*result_code*, *matched_dn=''*, *diagnostic_message=''*, *referral=None*, *server_sasl_credentials=None*)

ldap3.operation.bind.**bind_response_to_dict**(*response*)

ldap3.operation.bind.**bind_response_to_dict_fast**(*response*)

ldap3.operation.bind.**sicily_bind_response_to_dict**(*response*)

ldap3.operation.bind.**sicily_bind_response_to_dict_fast**(*response*)

### ldap3.operation.compare module

ldap3.operation.compare.**compare_operation**(*dn*, *attribute*, *value*, *auto_encode*, *schema=None*, *validator=None*)

ldap3.operation.compare.**compare_request_to_dict**(*request*)

ldap3.operation.compare.**compare_response_to_dict**(*response*)

### ldap3.operation.delete module

ldap3.operation.delete.**delete_operation**(*dn*)

ldap3.operation.delete.**delete_request_to_dict**(*request*)

ldap3.operation.delete.**delete_response_to_dict**(*response*)

### ldap3.operation.extended module

ldap3.operation.extended.**extended_operation**(*request_name*, *request_value=None*, *no_encode=None*)

ldap3.operation.extended.**extended_request_to_dict**(*request*)

ldap3.operation.extended.**extended_response_to_dict**(*response*)

ldap3.operation.extended.**extended_response_to_dict_fast**(*response*)

ldap3.operation.extended.**intermediate_response_to_dict**(*response*)

ldap3.operation.extended.**intermediate_response_to_dict_fast**(*response*)

### ldap3.operation.modify module

ldap3.operation.modify.**modify_operation**(*dn*, *changes*, *auto_encode*, *schema=None*, *validator=None*)

ldap3.operation.modify.**modify_request_to_dict**(*request*)

ldap3.operation.modify.**modify_response_to_dict**(*response*)

### ldap3.operation.modifyDn module

ldap3.operation.modifyDn.**modify_dn_operation**(*dn*, *new_relative_dn*, *delete_old_rdn=True*, *new_superior=None*)

ldap3.operation.modifyDn.**modify_dn_request_to_dict**(*request*)

ldap3.operation.modifyDn.**modify_dn_response_to_dict**(*response*)

### ldap3.operation.search module

**class** ldap3.operation.search.**FilterNode**(*tag=None*, *assertion=None*)
Bases: object

**append**(*filter_node*)

ldap3.operation.search.**attributes_to_dict**(*attribute_list*)

ldap3.operation.search.**attributes_to_dict_fast**(*attribute_list*)

ldap3.operation.search.**build_attribute_selection**(*attribute_list*, *schema*)

ldap3.operation.search.**checked_attributes_to_dict**(*attribute_list*, *schema=None*, *custom_formatter=None*)

ldap3.operation.search.**checked_attributes_to_dict_fast**(*attribute_list*, *schema=None*, *custom_formatter=None*)

ldap3.operation.search.**compile_filter**(*filter_node*)
Builds ASN1 structure for filter, converts from filter LDAP escaping to bytes

ldap3.operation.search.**decode_raw_vals**(*vals*)

ldap3.operation.search.**decode_raw_vals_fast**(*vals*)

ldap3.operation.search.**decode_vals**(*vals*)

ldap3.operation.search.**decode_vals_fast**(*vals*)

ldap3.operation.search.**evaluate_match**(*match*, *schema*, *auto_escape*, *auto_encode*)

ldap3.operation.search.**filter_to_string**(*filter_object*)

ldap3.operation.search.**matching_rule_assertion_to_string**(*matching_rule_assertion*)

ldap3.operation.search.**parse_filter**(*search_filter*, *schema*, *auto_escape*, *auto_encode*)

ldap3.operation.search.**raw_attributes_to_dict**(*attribute_list*)

ldap3.operation.search.**raw_attributes_to_dict_fast**(*attribute_list*)

ldap3.operation.search.**search_operation**(*search_base*, *search_filter*, *search_scope*, *dereference_aliases*, *attributes*, *size_limit*, *time_limit*, *types_only*, *auto_escape*, *auto_encode*, *schema=None*)

ldap3.operation.search.**search_request_to_dict**(*request*)

ldap3.operation.search.**search_result_done_response_to_dict**(*response*)

ldap3.operation.search.**search_result_entry_response_to_dict**(*response*, *schema*, *custom_formatter*, *check_names*)

ldap3.operation.search.**search_result_entry_response_to_dict_fast**(*response*, *schema*, *custom_formatter*, *check_names*)

ldap3.operation.search.**search_result_reference_response_to_dict**(*response*)

ldap3.operation.search.**search_result_reference_response_to_dict_fast**(*response*)

### ldap3.operation.unbind module

ldap3.operation.unbind.**unbind_operation**()

### Module contents

### ldap3.protocol package

### Subpackages

### ldap3.protocol.formatters package

### Submodules

### ldap3.protocol.formatters.formatters module

ldap3.protocol.formatters.formatters.**format_ad_timestamp**(*raw_value*)
Active Directory stores date/time values as the number of 100-nanosecond intervals that have elapsed since the 0 hour on January 1, 1601 till the date/time that is being stored. The time is always stored in Greenwich Mean Time (GMT) in the Active Directory.

ldap3.protocol.formatters.formatters.**format_binary**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_boolean**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_integer**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_sid**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_time**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_unicode**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_uuid**(*raw_value*)

ldap3.protocol.formatters.formatters.**format_uuid_le**(*raw_value*)

### ldap3.protocol.formatters.standard module

ldap3.protocol.formatters.standard.**find_attribute_helpers**(*attr_type*, *name*, *custom_formatter*)
Tries to format following the OIDs info and format_helper specification. Search for attribute oid, then attribute name (can be multiple), then attribute syntax Precedence is: 1. attribute name 2. attribute oid(from schema) 3. attribute names (from oid_info) 4. attribute syntax (from schema) Custom formatters can be defined in Server object and have precedence over the standard_formatters If no formatter is found the raw_value is returned as bytes. Attributes defined as SINGLE_VALUE in schema are returned as a single object, otherwise are returned as a list of object Formatter functions can return any kind of object return a tuple (formatter, validator)

ldap3.protocol.formatters.standard.**find_attribute_validator**(*schema*, *name*, *custom_validator*)

ldap3.protocol.formatters.standard.**format_attribute_values**(*schema*, *name*, *values*, *custom_formatter*)

### ldap3.protocol.formatters.validators module

ldap3.protocol.formatters.validators.**always_valid**(*input_value*)

ldap3.protocol.formatters.validators.**check_type**(*input_value*, *value_type*)

ldap3.protocol.formatters.validators.**validate_ad_timestamp**(*input_value*)
Active Directory stores date/time values as the number of 100-nanosecond intervals that have elapsed since the 0 hour on January 1, 1601 till the date/time that is being stored. The time is always stored in Greenwich Mean Time (GMT) in the Active Directory.

ldap3.protocol.formatters.validators.**validate_boolean**(*input_value*)

ldap3.protocol.formatters.validators.**validate_bytes**(*input_value*)

ldap3.protocol.formatters.validators.**validate_generic_single_value**(*input_value*)

ldap3.protocol.formatters.validators.**validate_integer**(*input_value*)

ldap3.protocol.formatters.validators.**validate_time**(*input_value*)

ldap3.protocol.formatters.validators.**validate_uuid**(*input_value*)
object guid in uuid format

ldap3.protocol.formatters.validators.**validate_uuid_le**(*input_value*)
Active Directory stores objectGUID in uuid_le format

### Module contents

### ldap3.protocol.sasl package

### Submodules

### ldap3.protocol.sasl.digestMd5 module

ldap3.protocol.sasl.digestMd5.**decode_directives**(*directives_string*)
converts directives to dict, unquote values

ldap3.protocol.sasl.digestMd5.**md5_h**(*value*)

ldap3.protocol.sasl.digestMd5.**md5_hex**(*value*)

ldap3.protocol.sasl.digestMd5.**md5_hmac**(*k*, *s*)

ldap3.protocol.sasl.digestMd5.**md5_kd**(*k*, *s*)

ldap3.protocol.sasl.digestMd5.**sasl_digest_md5**(*connection*, *controls*)

### ldap3.protocol.sasl.external module

ldap3.protocol.sasl.external.**sasl_external**(*connection*, *controls*)

### ldap3.protocol.sasl.kerberos module

### ldap3.protocol.sasl.plain module

ldap3.protocol.sasl.plain.**sasl_plain**(*connection*, *controls*)

### ldap3.protocol.sasl.sasl module

ldap3.protocol.sasl.sasl.**abort_sasl_negotiation**(*connection*, *controls*)

ldap3.protocol.sasl.sasl.**random_hex_string**(*size*)

ldap3.protocol.sasl.sasl.**sasl_prep**(*data*)
implement SASLPrep profile as per RFC4013: it defines the "SASLprep" profile of the "stringprep" algorithm [StringPrep]. The profile is designed for use in Simple Authentication and Security Layer ([SASL]) mechanisms, such as [PLAIN], [CRAM-MD5], and [DIGEST-MD5]. It may be applicable where simple user names and passwords are used. This profile is not intended for use in preparing identity strings that are not simple user names (e.g., email addresses, domain names, distinguished names), or where identity or password strings that are not character data, or require different handling (e.g., case folding).

ldap3.protocol.sasl.sasl.**send_sasl_negotiation**(*connection*, *controls*, *payload*)

ldap3.protocol.sasl.sasl.**validate_simple_password**(*password*, *accept_empty=False*)
validate simple password as per RFC4013 using sasl_prep:

### Module contents

### ldap3.protocol.schemas package

### Submodules

### ldap3.protocol.schemas.ad2012R2 module

### ldap3.protocol.schemas.ds389 module

### ldap3.protocol.schemas.edir888 module

### ldap3.protocol.schemas.slapd24 module

### Module contents

### Submodules

### ldap3.protocol.controls module

ldap3.protocol.controls.**build_control**(*oid*, *criticality*, *value*, *encode_control_value=True*)

### ldap3.protocol.convert module

ldap3.protocol.convert.**attribute_to_dict**(*attribute*)

ldap3.protocol.convert.**attributes_to_dict**(*attributes*)

ldap3.protocol.convert.**attributes_to_list**(*attributes*)

ldap3.protocol.convert.**authentication_choice_to_dict**(*authentication_choice*)

ldap3.protocol.convert.**ava_to_dict**(*ava*)

ldap3.protocol.convert.**build_controls_list**(*controls*)
controls is a list of Control() or tuples each tuple must have 3 elements: the control OID, the criticality, the value criticality must be a boolean

ldap3.protocol.convert.**change_to_dict**(*change*)

ldap3.protocol.convert.**changes_to_list**(*changes*)

ldap3.protocol.convert.**partial_attribute_to_dict**(*modification*)

ldap3.protocol.convert.**prepare_changes_for_request**(*changes*)

ldap3.protocol.convert.**prepare_filter_for_sending**(*raw_string*)

ldap3.protocol.convert.**prepare_for_sending**(*raw_string*)

ldap3.protocol.convert.**referrals_to_list**(*referrals*)

ldap3.protocol.convert.**sasl_to_dict**(*sasl*)

ldap3.protocol.convert.**search_refs_to_list**(*search_refs*)

ldap3.protocol.convert.**search_refs_to_list_fast**(*search_refs*)

ldap3.protocol.convert.**substring_to_dict**(*substring*)

ldap3.protocol.convert.**validate_assertion_value**(*schema*, *name*, *value*, *auto_escape*, *auto_encode*)

ldap3.protocol.convert.**validate_attribute_value**(*schema*, *name*, *value*, *auto_encode*, *validator=None*)

### ldap3.protocol.microsoft module

**class** ldap3.protocol.microsoft.**DirSyncControlRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('Flags', Integer()), NamedType('MaxBytes', Integer()), NamedType('Cook**

**class** ldap3.protocol.microsoft.**DirSyncControlResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('MoreResults', Integer()), NamedType('unused', Integer()), NamedType('**

**class** ldap3.protocol.microsoft.**ExtendedDN**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('option', Integer()))**

**class** ldap3.protocol.microsoft.**SicilyBindResponse**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('resultCode', ResultCode()), NamedType('serverCreds', OctetString()), Na**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=1))**

ldap3.protocol.microsoft.**dir_sync_control**(*criticality*, *object_security*, *ancestors_first*, *public_data_only*, *incremental_values*, *max_length*, *cookie*)

ldap3.protocol.microsoft.**extended_dn_control**(*criticality=False*, *hex_format=False*)

ldap3.protocol.microsoft.**show_deleted_control**(*criticality=False*)

### ldap3.protocol.novell module

**class** ldap3.protocol.novell.**CreateGroupTypeRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('createGroupType', LDAPOID()), OptionalNamedType('createGroupValu**

**class** ldap3.protocol.novell.**CreateGroupTypeResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('createGroupCookie', GroupCookie()), OptionalNamedType('createGroup**

**class** ldap3.protocol.novell.**EndGroupTypeRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('endGroupCookie', GroupCookie()), OptionalNamedType('endGroupValu**

**class** ldap3.protocol.novell.**EndGroupTypeResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(OptionalNamedType('endGroupValue', OctetString()))**

**class** ldap3.protocol.novell.**Error**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Integer

**tagSet = TagSet((), Tag(tagClass=0, tagFormat=0, tagId=2))**

**class** ldap3.protocol.novell.**GroupCookie**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Integer

**tagSet = TagSet((), Tag(tagClass=0, tagFormat=0, tagId=2))**

**class** ldap3.protocol.novell.**GroupingControlValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('groupingCookie', GroupCookie()), OptionalNamedType('groupValue', O**

**class** ldap3.protocol.novell.**Identity**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**class** ldap3.protocol.novell.**LDAPDN**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=0, tagFormat=0, tagId=4))**

**class** ldap3.protocol.novell.**LDAPOID**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=0, tagFormat=0, tagId=4))**

**class** ldap3.protocol.novell.**NmasGetUniversalPasswordRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('nmasver', NmasVer()), NamedType('reqdn', Identity()))**

**class** ldap3.protocol.novell.**NmasGetUniversalPasswordResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('nmasver', NmasVer()), NamedType('err', Error()), OptionalNamedType(**

class ldap3.protocol.novell.**NmasSetUniversalPasswordRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('nmasver', NmasVer()), NamedType('reqdn', Identity()), NamedType('nev**

class ldap3.protocol.novell.**NmasSetUniversalPasswordResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('nmasver', NmasVer()), NamedType('err', Error()))**

class ldap3.protocol.novell.**NmasVer**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Integer

**tagSet = TagSet((), Tag(tagClass=0, tagFormat=0, tagId=2))**

class ldap3.protocol.novell.**Password**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=0, tagFormat=0, tagId=4))**

class ldap3.protocol.novell.**ReplicaInfoRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('server_dn', LDAPDN()), NamedType('partition_dn', LDAPDN()))**

**tagSet = TagSet((), )**

class ldap3.protocol.novell.**ReplicaInfoResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('partition_id', Integer()), NamedType('replica_state', Integer()), NamedTy**

**tagSet = TagSet((), )**

class ldap3.protocol.novell.**ReplicaList**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = OctetString()**

### ldap3.protocol.oid module

ldap3.protocol.oid.**constant_to_oid_kind**(*oid_kind*)

ldap3.protocol.oid.**decode_oids**(*sequence*)

ldap3.protocol.oid.**decode_syntax**(*syntax*)

ldap3.protocol.oid.**oid_to_string**(*oid*)

### ldap3.protocol.persistentSearch module

class ldap3.protocol.persistentSearch.**ChangeType**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Enumerated

**namedValues = NamedValues((('modDN', 8), ('modify', 4), ('delete', 2), ('add', 1)))**

class ldap3.protocol.persistentSearch.**EntryChangeNotificationControl**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('changeType', ChangeType()), OptionalNamedType('previousDN', LDAP**

**class** ldap3.protocol.persistentSearch.**PersistentSearchControl**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('changeTypes', Integer()), NamedType('changesOnly', Boolean()), Named**

ldap3.protocol.persistentSearch.**persistent_search_control**(*change_types*, *changes_only=True*, *return_ecs=True*, *criticality=False*)

### ldap3.protocol.rfc2696 module

**class** ldap3.protocol.rfc2696.**Cookie**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**class** ldap3.protocol.rfc2696.**Integer0ToMax**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Integer

**subtypeSpec = ConstraintsIntersection(ValueRangeConstraint(0, Integer(2147483647)))**

**class** ldap3.protocol.rfc2696.**RealSearchControlValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('size', Size()), NamedType('cookie', Cookie()))**

**class** ldap3.protocol.rfc2696.**Size**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc2696.Integer0ToMax

ldap3.protocol.rfc2696.**paged_search_control**(*criticality=False*, *size=10*, *cookie=None*)

### ldap3.protocol.rfc2849 module

ldap3.protocol.rfc2849.**add_attributes**(*attributes*, *all_base64*)

ldap3.protocol.rfc2849.**add_controls**(*controls*, *all_base64*)

ldap3.protocol.rfc2849.**add_ldif_header**(*ldif_lines*)

ldap3.protocol.rfc2849.**add_request_to_ldif**(*entry*, *all_base64*, *sort_order=None*)

ldap3.protocol.rfc2849.**decode_persistent_search_control**(*change*)

ldap3.protocol.rfc2849.**delete_request_to_ldif**(*entry*, *all_base64*, *sort_order=None*)

ldap3.protocol.rfc2849.**ldif_sort**(*line*, *sort_order*)

ldap3.protocol.rfc2849.**modify_dn_request_to_ldif**(*entry*, *all_base64*, *sort_order=None*)

ldap3.protocol.rfc2849.**modify_request_to_ldif**(*entry*, *all_base64*, *sort_order=None*)

ldap3.protocol.rfc2849.**operation_to_ldif**(*operation_type*, *entries*, *all_base64=False*, *sort_order=None*)

ldap3.protocol.rfc2849.**persistent_search_response_to_ldif**(*change*)

ldap3.protocol.rfc2849.**safe_ldif_string**(*bytes_value*)

ldap3.protocol.rfc2849.**search_response_to_ldif**(*entries*, *all_base64*, *sort_order=None*)

ldap3.protocol.rfc2849.**sort_ldif_lines**(*lines*, *sort_order*)

### ldap3.protocol.rfc3062 module

**class** ldap3.protocol.rfc3062.**GenPasswd**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

newPasswd [2] OCTET STRING OPTIONAL

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

**class** ldap3.protocol.rfc3062.**NewPasswd**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

newPasswd [2] OCTET STRING OPTIONAL

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=2))**

**class** ldap3.protocol.rfc3062.**OldPasswd**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

oldPasswd [1] OCTET STRING OPTIONAL

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=1))**

**class** ldap3.protocol.rfc3062.**PasswdModifyRequestValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**PasswdModifyRequestValue ::= SEQUENCE {** userIdentity [0] OCTET STRING OPTIONAL oldPasswd [1] OCTET STRING OPTIONAL newPasswd [2] OCTET STRING OPTIONAL }

**componentType = NamedTypes(OptionalNamedType('userIdentity', UserIdentity()), OptionalNamedType('oldPasswd'**

**class** ldap3.protocol.rfc3062.**PasswdModifyResponseValue**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**PasswdModifyResponseValue ::= SEQUENCE {** genPasswd [0] OCTET STRING OPTIONAL }

**componentType = NamedTypes(OptionalNamedType('genPasswd', GenPasswd()))**

**class** ldap3.protocol.rfc3062.**UserIdentity**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

userIdentity [0] OCTET STRING OPTIONAL

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

### ldap3.protocol.rfc4511 module

**class** ldap3.protocol.rfc4511.**AbandonRequest**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.MessageID

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=0, tagId=16))**

**class** ldap3.protocol.rfc4511.**AddRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('entry', LDAPDN()), NamedType('attributes', AttributeList()))**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=8))**

**class** ldap3.protocol.rfc4511.**AddResponse**(***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPResult

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=9))**

**class** ldap3.protocol.rfc4511.**And**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SetOf

**componentType = Filter()**

**subtypeSpec = ConstraintsIntersection(ValueSizeConstraint(1, Integer(2147483647)))**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=0))**

**class** ldap3.protocol.rfc4511.**Any**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.AssertionValue

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=1))**

**class** ldap3.protocol.rfc4511.**ApproxMatch**(***kwargs*)
Bases: ldap3.protocol.rfc4511.AttributeValueAssertion

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=8))**

**class** ldap3.protocol.rfc4511.**AssertionValue**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**class** ldap3.protocol.rfc4511.**Attribute**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('type', AttributeDescription()), NamedType('vals', Vals()))**

**class** ldap3.protocol.rfc4511.**AttributeDescription**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPString

**class** ldap3.protocol.rfc4511.**AttributeList**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = Attribute()**

**class** ldap3.protocol.rfc4511.**AttributeSelection**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = Selector()**

**class** ldap3.protocol.rfc4511.**AttributeValue**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**class** ldap3.protocol.rfc4511.**AttributeValueAssertion**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('attributeDesc', AttributeDescription()), NamedType('assertionValue', Ass**

**class** ldap3.protocol.rfc4511.**AuthenticationChoice**(***kwargs*)
Bases: pyasn1.type.univ.Choice

**componentType = NamedTypes(NamedType('simple', Simple()), NamedType('sasl', SaslCredentials()), NamedType('sic**

**class** ldap3.protocol.rfc4511.**BindRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('version', Version()), NamedType('name', LDAPDN()), NamedType('auth**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=0))**

**class** ldap3.protocol.rfc4511.**BindResponse**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('resultCode', ResultCode()), NamedType('matchedDN', LDAPDN()), Nam**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=1))**

**class** ldap3.protocol.rfc4511.**Change**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('operation', Operation()), NamedType('modification', PartialAttribute()))**

**class** ldap3.protocol.rfc4511.**Changes**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = Change()**

**class** ldap3.protocol.rfc4511.**CompareRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('entry', LDAPDN()), NamedType('ava', AttributeValueAssertion()))**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=14))**

**class** ldap3.protocol.rfc4511.**CompareResponse**(***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPResult

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=15))**

**class** ldap3.protocol.rfc4511.**Control**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('controlType', LDAPOID()), DefaultedNamedType('criticality', Criticality**

**class** ldap3.protocol.rfc4511.**ControlValue**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**class** ldap3.protocol.rfc4511.**Controls**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = Control()**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=0))**

**class** ldap3.protocol.rfc4511.**Credentials**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**class** ldap3.protocol.rfc4511.**Criticality**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Boolean

**defaultValue = False**

**class** ldap3.protocol.rfc4511.**DelRequest**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPDN

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=0, tagId=10))**

**class** ldap3.protocol.rfc4511.**DelResponse**(***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPResult

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=11))**

**class** ldap3.protocol.rfc4511.**DeleteOldRDN**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Boolean

**class** ldap3.protocol.rfc4511.**DerefAliases**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Enumerated

**namedValues = NamedValues((('derefInSearching', 1), ('neverDerefAliases', 0), ('derefAlways', 3), ('derefFindingBaseO**

**class** ldap3.protocol.rfc4511.**DnAttributes**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Boolean

**defaultValue = Boolean('False')**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=4))**

**class** ldap3.protocol.rfc4511.**EqualityMatch**(***kwargs*)
Bases: ldap3.protocol.rfc4511.AttributeValueAssertion

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=3))**

**class** ldap3.protocol.rfc4511.**ExtendedRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('requestName', RequestName()), OptionalNamedType('requestValue', Re**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=23))**

**class** ldap3.protocol.rfc4511.**ExtendedResponse**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('resultCode', ResultCode()), NamedType('matchedDN', LDAPDN()), Nam**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=24))**

**class** ldap3.protocol.rfc4511.**ExtensibleMatch**(***kwargs*)
Bases: ldap3.protocol.rfc4511.MatchingRuleAssertion

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=9))**

**class** ldap3.protocol.rfc4511.**Filter**(***kwargs*)
Bases: pyasn1.type.univ.Choice

**componentType = NamedTypes(NamedType('and', And(componentType=None)), NamedType('or', Or(componentType**

**class** ldap3.protocol.rfc4511.**Final**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.AssertionValue

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=2))**

**class** ldap3.protocol.rfc4511.**GreaterOrEqual**(***kwargs*)
Bases: ldap3.protocol.rfc4511.AttributeValueAssertion

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=5))**

**class** ldap3.protocol.rfc4511.**Initial**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.AssertionValue

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

**class** ldap3.protocol.rfc4511.**Integer0ToMax**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Integer

**subtypeSpec = ConstraintsIntersection(ValueRangeConstraint(0, Integer(2147483647)))**

**class** ldap3.protocol.rfc4511.**IntermediateResponse**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(OptionalNamedType('responseName', IntermediateResponseName()), OptionalNamed**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=25))**

**class** ldap3.protocol.rfc4511.**IntermediateResponseName**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPOID

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

**class** ldap3.protocol.rfc4511.**IntermediateResponseValue**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=1))**

**class** ldap3.protocol.rfc4511.**LDAPDN**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPString

**class** ldap3.protocol.rfc4511.**LDAPMessage**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('messageID', MessageID()), NamedType('protocolOp', ProtocolOp()), Opt**

**class** ldap3.protocol.rfc4511.**LDAPOID**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**class** ldap3.protocol.rfc4511.**LDAPResult**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('resultCode', ResultCode()), NamedType('matchedDN', LDAPDN()), Nam**

**class** ldap3.protocol.rfc4511.**LDAPString**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**class** ldap3.protocol.rfc4511.**LessOrEqual**(***kwargs*)
Bases: ldap3.protocol.rfc4511.AttributeValueAssertion

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=6))**

**class** ldap3.protocol.rfc4511.**MatchValue**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.AssertionValue

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=3))**

**class** ldap3.protocol.rfc4511.**MatchingRule**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.MatchingRuleId

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=1))**

**class** ldap3.protocol.rfc4511.**MatchingRuleAssertion**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(OptionalNamedType('matchingRule', MatchingRule()), OptionalNamedType('type', T**

**class** ldap3.protocol.rfc4511.**MatchingRuleId**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPString

**class** ldap3.protocol.rfc4511.**MessageID**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.Integer0ToMax

**class** ldap3.protocol.rfc4511.**ModifyDNRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('entry', LDAPDN()), NamedType('newrdn', RelativeLDAPDN()), NamedT**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=12))**

**class** ldap3.protocol.rfc4511.**ModifyDNResponse**(***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPResult

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=13))**

**class** ldap3.protocol.rfc4511.**ModifyRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('object', LDAPDN()), NamedType('changes', Changes()))**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=6))**

**class** ldap3.protocol.rfc4511.**ModifyResponse**(***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPResult

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=7))**

**class** ldap3.protocol.rfc4511.**NewSuperior**(*value=NoValue(), **kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPDN

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

**class** ldap3.protocol.rfc4511.**Not**(***kwargs*)
Bases: pyasn1.type.univ.Choice

**componentType = NamedTypes(NamedType('innerNotFilter', Filter()))**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=2))**

**class** ldap3.protocol.rfc4511.**Operation**(*value=NoValue(), **kwargs*)
Bases: pyasn1.type.univ.Enumerated

**namedValues = NamedValues((('increment', 3), ('replace', 2), ('delete', 1), ('add', 0)))**

**class** ldap3.protocol.rfc4511.**Or**(**args, **kwargs*)
Bases: pyasn1.type.univ.SetOf

**componentType = Filter()**

**subtypeSpec = ConstraintsIntersection(ValueSizeConstraint(1, Integer(2147483647)))**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=1))**

**class** ldap3.protocol.rfc4511.**PartialAttribute**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('type', AttributeDescription()), NamedType('vals', Vals()))**

**class** ldap3.protocol.rfc4511.**PartialAttributeList**(**args, **kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = PartialAttribute()**

**class** ldap3.protocol.rfc4511.**Present**(*value=NoValue(), **kwargs*)
Bases: ldap3.protocol.rfc4511.AttributeDescription

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=7))**

**class** ldap3.protocol.rfc4511.**ProtocolOp**(***kwargs*)
Bases: pyasn1.type.univ.Choice

**componentType = NamedTypes(NamedType('bindRequest', BindRequest()), NamedType('bindResponse', BindRespons**

class ldap3.protocol.rfc4511.**Referral**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = URI()**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=3))**

class ldap3.protocol.rfc4511.**RelativeLDAPDN**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPString

class ldap3.protocol.rfc4511.**RequestName**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPOID

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

class ldap3.protocol.rfc4511.**RequestValue**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=1))**

class ldap3.protocol.rfc4511.**ResponseName**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPOID

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=10))**

class ldap3.protocol.rfc4511.**ResponseValue**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=11))**

class ldap3.protocol.rfc4511.**ResultCode**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Enumerated

**namedValues = NamedValues((('protocolError', 2), ('timeLimitExceeded', 3), ('assertionFailed', 122), ('sizeLimitExcee**

**subTypeSpec = ConstraintsIntersection(SingleValueConstraint(0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20,**

class ldap3.protocol.rfc4511.**SaslCredentials**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('mechanism', LDAPString()), OptionalNamedType('credentials', Credenti**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=3))**

class ldap3.protocol.rfc4511.**Scope**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Enumerated

**namedValues = NamedValues((('baseObject', 0), ('wholeSubtree', 2), ('singleLevel', 1)))**

class ldap3.protocol.rfc4511.**SearchRequest**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('baseObject', LDAPDN()), NamedType('scope', Scope()), NamedType('de**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=3))**

class ldap3.protocol.rfc4511.**SearchResultDone**(***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPResult

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=5))**

**class** ldap3.protocol.rfc4511.**SearchResultEntry**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('object', LDAPDN()), NamedType('attributes', PartialAttributeList()))**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=4))**

**class** ldap3.protocol.rfc4511.**SearchResultReference**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = URI()**

**subtypeSpec = ConstraintsIntersection(ValueSizeConstraint(1, Integer(2147483647)))**

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=32, tagId=19))**

**class** ldap3.protocol.rfc4511.**Selector**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPString

**class** ldap3.protocol.rfc4511.**ServerSaslCreds**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=7))**

**class** ldap3.protocol.rfc4511.**SicilyNegotiate**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=10))**

**class** ldap3.protocol.rfc4511.**SicilyPackageDiscovery**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=9))**

**class** ldap3.protocol.rfc4511.**SicilyResponse**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=11))**

**class** ldap3.protocol.rfc4511.**Simple**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.OctetString

**encoding = 'utf-8'**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=0))**

**class** ldap3.protocol.rfc4511.**Substring**(***kwargs*)
Bases: pyasn1.type.univ.Choice

**componentType = NamedTypes(NamedType('initial', Initial()), NamedType('any', Any()), NamedType('final', Final()))**

**class** ldap3.protocol.rfc4511.**SubstringFilter**(***kwargs*)
Bases: pyasn1.type.univ.Sequence

**componentType = NamedTypes(NamedType('type', AttributeDescription()), NamedType('substrings', Substrings()))**

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=32, tagId=4))**

**class** ldap3.protocol.rfc4511.**Substrings**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SequenceOf

**componentType = Substring()**

**subtypeSpec = ConstraintsIntersection(ValueSizeConstraint(1, Integer(2147483647)))**

**class** ldap3.protocol.rfc4511.**Type**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.AttributeDescription

**tagSet = TagSet((), Tag(tagClass=128, tagFormat=0, tagId=2))**

**class** ldap3.protocol.rfc4511.**TypesOnly**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Boolean

**class** ldap3.protocol.rfc4511.**URI**(*value=NoValue()*, ***kwargs*)
Bases: ldap3.protocol.rfc4511.LDAPString

**class** ldap3.protocol.rfc4511.**UnbindRequest**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Null

**tagSet = TagSet((), Tag(tagClass=64, tagFormat=0, tagId=2))**

**class** ldap3.protocol.rfc4511.**Vals**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SetOf

**componentType = AttributeValue()**

**class** ldap3.protocol.rfc4511.**ValsAtLeast1**(**args*, ***kwargs*)
Bases: pyasn1.type.univ.SetOf

**componentType = AttributeValue()**

**subtypeSpec = ConstraintsIntersection(ValueSizeConstraint(1, Integer(2147483647)))**

**class** ldap3.protocol.rfc4511.**Version**(*value=NoValue()*, ***kwargs*)
Bases: pyasn1.type.univ.Integer

**subtypeSpec = ConstraintsIntersection(ValueRangeConstraint(1, 127))**

### ldap3.protocol.rfc4512 module

**class** ldap3.protocol.rfc4512.**AttributeTypeInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *superior=None*, *equality=None*, *ordering=None*, *substring=None*, *syntax=None*, *min_length=None*, *single_value=False*, *collective=False*, *no_user_modification=False*, *usage=None*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.2)

**class** ldap3.protocol.rfc4512.**BaseObjectInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: object

Base class for objects defined in the schema as per RFC4512

**classmethod from_definition**(*definitions*)

**oid_info**

**class** ldap3.protocol.rfc4512.**BaseServerInfo**(*raw_attributes*)
Bases: object

**classmethod from_file**(*target*, *schema=None*, *custom_formatter=None*)

**classmethod from_json**(*json_definition*, *schema=None*, *custom_formatter=None*)

**to_file**(*target*, *indent=4*, *sort=True*)

**to_json**(*indent=4*, *sort=True*)

**class** ldap3.protocol.rfc4512.**DitContentRuleInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *auxiliary_classes=None*, *must_contain=None*, *may_contain=None*, *not_contains=None*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.6)

**class** ldap3.protocol.rfc4512.**DitStructureRuleInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *name_form=None*, *superior=None*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.7.1)

**class** ldap3.protocol.rfc4512.**DsaInfo**(*attributes*, *raw_attributes*)
Bases: ldap3.protocol.rfc4512.BaseServerInfo

This class contains info about the ldap server (DSA) read from DSE as defined in RFC4512 and RFC3045. Unknown attributes are stored in the "other" dict

**class** ldap3.protocol.rfc4512.**LdapSyntaxInfo**(*oid=None*, *description=None*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.5)

**class** ldap3.protocol.rfc4512.**MatchingRuleInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *syntax=None*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.3)

**class** ldap3.protocol.rfc4512.**MatchingRuleUseInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *apply_to=None*, *extensions=None*, *experimental=None*, *definition=None*)
Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.4)

**class** ldap3.protocol.rfc4512.**NameFormInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *object_class=None*, *must_contain=None*, *may_contain=None*, *extensions=None*, *experimental=None*, *definition=None*)

Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.7.2)

**class** ldap3.protocol.rfc4512.**ObjectClassInfo**(*oid=None*, *name=None*, *description=None*, *obsolete=False*, *superior=None*, *kind=None*, *must_contain=None*, *may_contain=None*, *extensions=None*, *experimental=None*, *definition=None*)

Bases: ldap3.protocol.rfc4512.BaseObjectInfo

As per RFC 4512 (4.1.1)

**class** ldap3.protocol.rfc4512.**SchemaInfo**(*schema_entry*, *attributes*, *raw_attributes*)

Bases: ldap3.protocol.rfc4512.BaseServerInfo

This class contains info about the ldap server schema read from an entry (default entry is DSE) as defined in RFC4512. Unknown attributes are stored in the "other" dict

**is_valid**()

ldap3.protocol.rfc4512.**attribute_usage_to_constant**(*value*)

ldap3.protocol.rfc4512.**constant_to_attribute_usage**(*value*)

ldap3.protocol.rfc4512.**constant_to_class_kind**(*value*)

ldap3.protocol.rfc4512.**extension_to_tuple**(*extension_string*)

ldap3.protocol.rfc4512.**list_to_string**(*list_object*)

ldap3.protocol.rfc4512.**oids_string_to_list**(*oid_string*)

ldap3.protocol.rfc4512.**quoted_string_to_list**(*quoted_string*)

### ldap3.protocol.rfc4527 module

ldap3.protocol.rfc4527.**post_read_control**(*attributes*, *criticality=False*)

Create a post-read control for a request. When passed as a control to the controls parameter of an operation, it will return the value in *Connection.result* after the operation took place.

ldap3.protocol.rfc4527.**pre_read_control**(*attributes*, *criticality=False*)

Create a pre-read control for a request. When passed as a control to the controls parameter of an operation, it will return the value in *Connection.result* before the operation took place.

### Module contents

### ldap3.strategy package

### Submodules

### ldap3.strategy.async module

**class** ldap3.strategy.async.**AsyncStrategy**(*ldap_connection*)

Bases: ldap3.strategy.base.BaseStrategy

This strategy is asynchronous. You send the request and get the messageId of the request sent Receiving data from socket is managed in a separated thread in a blocking mode Requests return an int value to indicate the messageId of the requested Operation You get the response with get_response, it has a timeout to wait for

response to appear Connection.response will contain the whole LDAP response for the messageId requested in a dict form Connection.request will contain the result LDAP message in a dict form Response appear in strategy._responses dictionary

**class** **ReceiverSocketThread**(*ldap_connection*)
Bases: threading.Thread

The thread that actually manage the receiver socket

**run**()
Wait for data on socket, compute the length of the message and wait for enough bytes to decode the message Message are appended to strategy._responses

AsyncStrategy.**close**()
Close connection and stop socket thread

AsyncStrategy.**get_stream**()

AsyncStrategy.**open**(*reset_usage=True*, *read_server_info=True*)
Open connection and start listen on the socket in a different thread

AsyncStrategy.**post_send_search**(*message_id*)
Clears connection.response and returns messageId

AsyncStrategy.**post_send_single_response**(*message_id*)
Clears connection.response and returns messageId.

AsyncStrategy.**receiving**()

AsyncStrategy.**set_stream**(*value*)

### ldap3.strategy.asyncStream module

**class** ldap3.strategy.asyncStream.**AsyncStreamStrategy**(*ldap_connection*)
Bases: ldap3.strategy.async.AsyncStrategy

This strategy is asynchronous. It streams responses in a generator as they appear in the self._responses container

**accumulate_stream**(*message_id*, *change*)

**get_stream**()

**set_stream**(*value*)

### ldap3.strategy.base module

**class** ldap3.strategy.base.**BaseStrategy**(*ldap_connection*)
Bases: object

Base class for connection strategy

**close**()
Close connection

**static** **compute_ldap_message_size**(*data*)
Compute LDAP Message size according to BER definite length rules Returns -1 if too few data to compute message length

**static decode_control**(*control*)
decode control, return a 2-element tuple where the first element is the control oid and the second element is a dictionary with description (from Oids), criticality and decoded control value

**static decode_control_fast**(*control*)
decode control, return a 2-element tuple where the first element is the control oid and the second element is a dictionary with description (from Oids), criticality and decoded control value

**static decode_request**(*message_type*, *component*, *controls=None*)

**decode_response**(*ldap_message*)
Convert received LDAPMessage to a dict

**decode_response_fast**(*ldap_message*)
Convert received LDAPMessage from fast ber decoder to a dict

**do_next_range_search**(*request*, *response*, *attr_name*)

**do_operation_on_referral**(*request*, *referrals*)

**do_search_on_auto_range**(*request*, *response*)

**get_response**(*message_id*, *timeout=None*, *get_request=False*)
Get response LDAP messages Responses are returned by the underlying connection strategy Check if message_id LDAP message is still outstanding and wait for timeout to see if it appears in _get_response Result is stored in connection.result Responses without result is stored in connection.response A tuple (responses, result) is returned

**get_stream**()

**open**(*reset_usage=True*, *read_server_info=True*)
Open a socket to a server. Choose a server from the server pool if available

**post_send_search**(*message_id*)

**post_send_single_response**(*message_id*)

**receiving**()

**send**(*message_type*, *request*, *controls=None*)
Send an LDAP message Returns the message_id

**sending**(*ldap_message*)

**set_stream**(*value*)

**unbind_referral_cache**()

**valid_referral_list**(*referrals*)

### ldap3.strategy.ldifProducer module

**class** ldap3.strategy.ldifProducer.**LdifProducerStrategy**(*ldap_connection*)
Bases: ldap3.strategy.base.BaseStrategy

This strategy is used to create the LDIF stream for the Add, Delete, Modify, ModifyDn operations. You send the request and get the request in the ldif-change representation of the operation. NO OPERATION IS SENT TO THE LDAP SERVER! Connection.request will contain the result LDAP message in a dict form Connection.response will contain the ldif-change format of the requested operation if available You don't need a real server to connect to for this strategy

**accumulate_stream**(*fragment*)

**get_stream**()

**post_send_search**(*message_id*)

**post_send_single_response**(*message_id*)

**receiving**()

**send**(*message_type*, *request*, *controls=None*)
Build the LDAPMessage without sending to server

**set_stream**(*value*)

### ldap3.strategy.mockAsync module

**class** ldap3.strategy.mockAsync.**MockAsyncStrategy**(*ldap_connection*)
Bases: ldap3.strategy.mockBase.MockBaseStrategy, ldap3.strategy.async.AsyncStrategy

This strategy create a mock LDAP server, with asynchronous access It can be useful to test LDAP without accessing a real Server

**get_response**(*message_id*, *timeout=None*, *get_request=False*)

**post_send_search**(*payload*)

**post_send_single_response**(*payload*)

### ldap3.strategy.mockBase module

**class** ldap3.strategy.mockBase.**MockBaseStrategy**
Bases: object

Base class for connection strategy

**add_entry**(*dn*, *attributes*)

**entries_from_json**(*json_entry_file*)

**equal**(*dn*, *attribute*, *value*)

**evaluate_filter_node**(*node*, *candidates*)
After evaluation each 2 sets are added to each MATCH node, one for the matched object and one for unmatched object. The unmatched object set is needed if a superior node is a NOT that reverts the evaluation. The BOOLEAN nodes mix the sets returned by the MATCH nodes

**mock_add**(*request_message*, *controls*)

**mock_bind**(*request_message*, *controls*)

**mock_compare**(*request_message*, *controls*)

**mock_delete**(*request_message*, *controls*)

**mock_extended**(*request_message*, *controls*)

**mock_modify**(*request_message*, *controls*)

**mock_modify_dn**(*request_message*, *controls*)

**mock_search**(*request_message*, *controls*)

**remove_entry**(*dn*)

**send**(*message_type*, *request*, *controls=None*)

**class** ldap3.strategy.mockBase.**PagedSearchSet**(*response*, *size*, *criticality*)
Bases: object

**next**(*size=None*)

ldap3.strategy.mockBase.**random_cookie**()

### ldap3.strategy.mockSync module

**class** ldap3.strategy.mockSync.**MockSyncStrategy**(*ldap_connection*)
Bases: ldap3.strategy.mockBase.MockBaseStrategy, ldap3.strategy.sync.SyncStrategy

This strategy create a mock LDAP server, with synchronous access It can be useful to test LDAP without accessing a real Server

**post_send_search**(*payload*)

**post_send_single_response**(*payload*)

### ldap3.strategy.restartable module

**class** ldap3.strategy.restartable.**RestartableStrategy**(*ldap_connection*)
Bases: ldap3.strategy.sync.SyncStrategy

**get_stream**()

**open**(*reset_usage=False*, *read_server_info=True*)

**post_send_search**(*message_id*)

**post_send_single_response**(*message_id*)

**send**(*message_type*, *request*, *controls=None*)

**set_stream**(*value*)

### ldap3.strategy.reusable module

**class** ldap3.strategy.reusable.**ReusableStrategy**(*ldap_connection*)
Bases: ldap3.strategy.base.BaseStrategy

A pool of reusable SyncWaitRestartable connections with lazy behaviour and limited lifetime. The connection using this strategy presents itself as a normal connection, but internally the strategy has a pool of connections that can be used as needed. Each connection lives in its own thread and has a busy/available status. The strategy performs the requested operation on the first available connection. The pool of connections is instantiated at strategy initialization. Strategy has two customizable properties, the total number of connections in the pool and the lifetime of each connection. When lifetime is expired the connection is closed and will be open again when needed.

**class ConnectionPool**(*connection*)
Bases: object

Container for the Connection Threads

**create_pool**()

**get_info_from_server**()

**rebind_pool**()

**start_pool**()

**terminate_pool**()

**class** ReusableStrategy.**PooledConnectionThread**(*worker*, *master_connection*)
Bases: threading.Thread

The thread that holds the Reusable connection and receive operation request via the queue Result are sent back in the pool._incoming list when ready

**run**()

**class** ReusableStrategy.**PooledConnectionWorker**(*connection*, *request_queue*)
Bases: object

Container for the restartable connection. it includes a thread and a lock to execute the connection in the pool

**new_connection**()

ReusableStrategy.**get_response**(*counter*, *timeout=None*, *get_request=False*)

ReusableStrategy.**get_stream**()

ReusableStrategy.**open**(*reset_usage=True*, *read_server_info=True*)

ReusableStrategy.**pools** = {}

ReusableStrategy.**post_send_search**(*counter*)

ReusableStrategy.**post_send_single_response**(*counter*)

ReusableStrategy.**receiving**()

ReusableStrategy.**send**(*message_type*, *request*, *controls=None*)

ReusableStrategy.**set_stream**(*value*)

ReusableStrategy.**terminate**()

ReusableStrategy.**validate_bind**(*controls*)

### ldap3.strategy.sync module

**class** ldap3.strategy.sync.**SyncStrategy**(*ldap_connection*)
Bases: ldap3.strategy.base.BaseStrategy

This strategy is synchronous. You send the request and get the response Requests return a boolean value to indicate the result of the requested Operation Connection.response will contain the whole LDAP response for the messageId requested in a dict form Connection.request will contain the result LDAP message in a dict form

**get_stream**()

**open**(*reset_usage=True*, *read_server_info=True*)

**post_send_search**(*message_id*)
Executed after a search request Returns the result message and store in connection.response the objects found

**post_send_single_response**(*message_id*)
Executed after an Operation Request (except Search) Returns the result message or None

**receiving**()
Receive data over the socket Checks if the socket is closed

**set_stream**(*value*)

### Module contents

### ldap3.utils package

### Submodules

### ldap3.utils.asn1 module

**class** ldap3.utils.asn1.**BooleanCEREncoder**
Bases: pyasn1.codec.ber.encoder.BooleanEncoder

ldap3.utils.asn1.**compute_ber_size**(*data*)
Compute size according to BER definite length rules Returns size of value and value offset

ldap3.utils.asn1.**decode_bind_response**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_boolean**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_controls**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_extended_response**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_integer**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_intermediate_response**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_message_fast**(*message*)

ldap3.utils.asn1.**decode_octet_string**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**decode_sequence**(*message*, *start*, *stop*, *context_decoders=None*)

ldap3.utils.asn1.**get_byte**(*x*)

ldap3.utils.asn1.**get_bytes**(*x*)

ldap3.utils.asn1.**ldap_result_to_dict_fast**(*response*)

### ldap3.utils.ciDict module

**class** ldap3.utils.ciDict.**CaseInsensitiveDict**(*other=None*, ***kwargs*)
Bases: collections.abc.MutableMapping

**copy**()

**items**()

**keys**()

**values**()

**class** ldap3.utils.ciDict.**CaseInsensitiveWithAliasDict**(*other=None*, ***kwargs*)
Bases: ldap3.utils.ciDict.CaseInsensitiveDict

**aliases**()

**copy**()

**remove_alias**(*alias*)

**set_alias**(*key*, *alias*)

### ldap3.utils.config module

ldap3.utils.config.**get_config_parameter**(*parameter*)

ldap3.utils.config.**set_config_parameter**(*parameter*, *value*)

### ldap3.utils.conv module

ldap3.utils.conv.**check_json_dict**(*json_dict*)

ldap3.utils.conv.**escape_bytes**(*bytes_value*)
Convert a byte sequence to a properly escaped for LDAP (format BACKSLASH HEX HEX) string

ldap3.utils.conv.**escape_filter_chars**(*text*, *encoding=None*)
Escape chars mentioned in RFC4515.

ldap3.utils.conv.**format_json**(*obj*)

ldap3.utils.conv.**is_filter_escaped**(*text*)

ldap3.utils.conv.**json_encode_b64**(*obj*)

ldap3.utils.conv.**json_hook**(*obj*)

ldap3.utils.conv.**prepare_for_stream**(*value*)

ldap3.utils.conv.**to_raw**(*obj*, *encoding='utf-8'*)
Tries to convert to raw bytes from unicode

ldap3.utils.conv.**to_unicode**(*obj*, *encoding=None*, *from_server=False*)
Try to convert bytes (and str in python2) to unicode. Return object unmodified if python3 string, else raise an exception

### ldap3.utils.dn module

ldap3.utils.dn.**parse_dn**(*dn*, *escape=False*, *strip=True*)

ldap3.utils.dn.**safe_dn**(*dn*, *decompose=False*, *reverse=False*)
normalize and escape a dn, if dn is a sequence it is joined. the reverse parameter change the join direction of the sequence

ldap3.utils.dn.**safe_rdn**(*dn*, *decompose=False*)
Returns a list of rdn for the dn, usually there is only one rdn, but it can be more than one when the + sign is used

ldap3.utils.dn.**to_dn**(*iterator*, *decompose=False*, *remove_space=False*, *space_around_equal=False*, *separate_rdn=False*)
Convert an iterator to a list of dn parts if decompose=True return a list of tuple (one for each dn component) else return a list of strings if remove_space=True removes unneeded spaces if space_around_equal=True add spaces around equal in returned strings if separate_rdn=True consider multiple RDNs as different component of DN

### ldap3.utils.hashed module

ldap3.utils.hashed.**hashed**(*algorithm*, *value*, *salt=None*, *raw=False*, *encoding='utf-8'*)

### ldap3.utils.log module

ldap3.utils.log.**format_ldap_message**(*message*, *prefix*)

ldap3.utils.log.**get_detail_level_name**(*level_name*)

ldap3.utils.log.**get_library_log_activation_lavel**()

ldap3.utils.log.**get_library_log_detail_level**()

ldap3.utils.log.**get_library_log_hide_sensitive_data**()

ldap3.utils.log.**get_library_log_max_line_length**()

ldap3.utils.log.**log**(*detail*, *message*, **args*)

ldap3.utils.log.**log_enabled**(*detail*)

ldap3.utils.log.**set_library_log_activation_level**(*logging_level*)

ldap3.utils.log.**set_library_log_detail_level**(*detail*)

ldap3.utils.log.**set_library_log_hide_sensitive_data**(*hide=True*)

ldap3.utils.log.**set_library_log_max_line_length**(*length*)

### ldap3.utils.ntlm module

**class** ldap3.utils.ntlm.**NtlmClient**(*domain*, *user_name*, *password*)

Bases: object

**compute_nt_response**()

**create_authenticate_message**()

Microsoft MS-NLMP 2.2.1.3

**create_negotiate_message**()

Microsoft MS-NLMP 2.2.1.1

**get_client_flag**(*flag*)

**get_negotiated_flag**(*flag*)

**get_server_av_flag**(*flag*)

**ntowf_v2**()

**static pack_av_info**(*avs*)

**static pack_field**(*value*, *offset*)

**static pack_windows_timestamp**()

**parse_challenge_message**(*message*)

Microsoft MS-NLMP 2.2.1.2

**reset_client_flags**()

**set_client_flag**(*flags*)

**static unpack_av_info**(*info*)

**static unpack_field**(*field_message*)

**unset_client_flag**(*flags*)

ldap3.utils.ntlm.**pack_windows_version**(*debug=False*)

ldap3.utils.ntlm.**unpack_windows_version**(*version_message*)

### ldap3.utils.ordDict module

### ldap3.utils.repr module

ldap3.utils.repr.**to_stdout_encoding**(*value*)

### ldap3.utils.tls_backport module

**exception** ldap3.utils.tls_backport.**CertificateError**
Bases: ValueError

ldap3.utils.tls_backport.**match_hostname**(*cert*, *hostname*)
Backported from Python 3.4.3 standard library.

Verify that *cert* (in decoded format as returned by SSLSocket.getpeercert()) matches the *hostname*. RFC 2818 and RFC 6125 rules are followed, but IP addresses are not accepted for *hostname*.

CertificateError is raised on failure. On success, the function returns nothing.

### ldap3.utils.uri module

ldap3.utils.uri.**parse_uri**(*uri*)
Decode LDAP URI as specified in RFC 4516 relaxing specifications permitting 'ldaps' as scheme for ssl-ldap

### Module contents

### Submodules

### ldap3.version module

### Module contents

CHAPTER 2

# Indices and tables

- genindex
- modindex

# Python Module Index

## l

# Index

## A

## B



## C

## D

## E

## F

## G



## H



## I

## M

## N



## O



## P

## Q



## R

## S

## T

## U



## V

## W