ezldap Documentation

Release 0.6

Jeff Stafford

Aug 15, 2018

Getting started

ezldap is an object-oriented Python LDAP API and command-line LDAP client to make working with your directory server as fast and easy as possible.

Although several very comprehensive LDAP API's currently exist (notably python-ldap and ldap3), neither are particularly easy to use and generally focus more on the specifics of communicating via LDAP rather than exposing an easy to use high-level API. This package is designed to do that. The target audience of this package is system administrators and support staff who are on a timeline and just need to add a user or write a quick script in a portable manner.

So how is ezldap different? It is a wrapper around the ldap3 API that gives access to easily perform a number of high-level tasks like searching an LDAP directory, adding a user and sending them an email, or resetting someone's password securely. This process is streamlined by a configuration script that autodetects as many LDAP configuration values as possible and automates future connections to the directory server. Additionally, additions to an LDAP directory are performed using a set of configurable LDIF templates, making it very quick to customize the behavior of certain actions (like adding a group). Finally, for most tasks, a command-line interface is provided, for users who just want to get started doing stuff.

Python example:

```
import ezldap

with ezldap.auto_bind() as con:
    con.add_user('username', 'groupname', 'password')
```

Command-line example:

```
# a password will be automatically generated
ezldap add_user username groupname
```

For a quick tour of what ezldap can do, why not check out the package walkthrough?

CHAPTER 1

Installation

ezldap has no dependencies aside from any currently supported version of Python 3. ezldap is tested against all current versions of Python 3 (3.4, 3.5, and 3.6). To install ezldap, just use pip:

```
pip install ezldap
ezldap config
```

1.1 Working with the development version

To install the development version from Github, use the following command: It is *highly* suggested to run the tests if you will be working with the development version.

```
# install github release
pip install git+https://github.com/jstaf/ezldap.git
ezldap config

# to run tests
pip install pytest pytest-docker pytest-cov docker-compose
pytest
```

CHAPTER 2

Configuration

2.1 Configure ezldap before use

Both the Python API and the command line client use a set of config values and LDIF templates stored in `~/.ezldap/`. To create these configs, run the following command:

```
ezldap config
```

Sample output (default values are in brackets, just press `Enter` to accept the defaults and move to the next option):

```
LDAP host [ldap:///]:
Bind DN (leave blank for anonymous bind) [cn=Manager,dc=ezldap,dc=io]:
Bind password (leave blank to prompt for password) [password]:
User base dn [ou=People,dc=ezldap,dc=io]:
Group base dn [ou=Group,dc=ezldap,dc=io]:
Host base dn [ou=Hosts,dc=ezldap,dc=io]:
Default home directory for new users [/home]:
```

2.2 Delete your ezldap configuration

To delete an ezldap configuration and start from a clean slate, it's as easy as:

```
rm -r ~/.ezldap
```

CHAPTER 3

Walkthrough

This is a quick walkthrough designed to show the basics of using ezldap to query and modify an LDAP directory. Please install Docker before beginning this tutorial.

3.1 Spin up a test LDAP server

We will begin by starting an example LDAP server (OpenLDAP, in this case) to add and subtract objects to. This essentially is a throw-away LDAP server that will not impact our computer or a production environment. You can test out operations on this container to your heart's content.

To start our example LDAP server:

```
docker run -p 389:389 -p 636:636 jstaf/ezldap
```

You should see something like the following:

```
5b1b49a6 @(#) $OpenLDAP: slapd 2.4.45 (Dec  6 2017 14:25:36) $
  mockbuild@buildhw-08.phx2.fedoraproject.org:/builddir/build/BUILD/openldap-2.4.45/
↪openldap-2.4.45/servers/slapd
5b1b49a6 slapd starting
```

This is an example LDAP server with debug logging on. As we make queries against this server, we will see them appear here. Though the actual content is not important, you can use this information to verify exactly what's happening when we perform LDAP operations later.

To stop the container later on, just use `Control-c`. You do not need to worry about cleaning up or revisiting this container later, it is completely disposable.

3.2 Setup and configure ezldap

To install ezldap, run the following:

```
pip install ezldap
```

Now we will configure ezldap to connect to the OpenLDAP instance running on our demonstration Docker container. Run `ezldap config`. You will be prompted for the following. I have provided the bind information for the container here:

- **LDAP host:** ldap:///
- **Bind DN:** cn=Manager,dc=ezldap,dc=io
- **Bind password:** password (yes, it's "password")
- **User base dn:** ou=People,dc=ezldap,dc=io
- **Group base dn:** ou=Group,dc=ezldap,dc=io
- **Host base dn:** ou=Hosts,dc=ezldap,dc=io
- **Default home directory:** /home

When running `ezldap config`, this will look like the following:

```
Configuring ezldap...
Default values are in [brackets] - to accept, press Enter.

LDAP host: ldap:///
Bind DN (leave blank for anonymous bind) [cn=Manager,dc=example,dc=com]: cn=Manager,
↪dc=ezldap,dc=io
Bind password (leave blank to prompt for password): password
User base dn [ou=People,dc=example,dc=com]: ou=People,dc=ezldap,dc=io
Group base dn [ou=Group,dc=example,dc=com]: ou=Group,dc=ezldap,dc=io
Host base dn [ou=Hosts,dc=example,dc=com]: ou=Hosts,dc=ezldap,dc=io
Default home directory for new users [/home]:

Writing configs to ~/.ezldap/
Edit config.yaml and the LDIF templates in ~/.ezldap/ to configure ezldap's behavior.
```

To check that you've entered this information correctly, you can run the `bind_info` command:

```
ezldap bind_info
```

If the Docker container is running and ezldap has been setup and configured correctly, you should see the following:

```
ldap://localhost:389 - cleartext - user: cn=Manager,dc=ezldap,dc=io - not lazy -␣
↪bound - open - <local: [::1]:36788 - remote: [::1]:389> - tls started - listening -␣
↪SyncStrategy - internal decoder
```

3.3 Next steps

Assuming you've reached this point, congratulations! You are now set to use ezldap.

- To get started using the command-line LDAP client, see the Bash recipes page, and run through the examples there.
- To get started with the Python API, go to the Python recipes page. More detailed information can be found by reading the Python API documentation.

CHAPTER 4

Bash recipes

ezldap provides a command-line client that attempts to mimic the Python API as closely as possible. The goal is to make common LDAP operations available via the command-line.

4.1 Query an LDAP server

4.1.1 Get information about your LDAP server

Every LDAP server will supply information about itself once connected. To fetch identity information from your LDAP server, you can use the `server_info` command:

```
ezldap server_info
```

Sample output:

```
DSA info (from DSE):
  Supported LDAP versions: 3
  Naming contexts:
    dc=ezldap,dc=io
  Supported controls:
    1.2.826.0.1.3344810.2.3 - Matched Values - Control - RFC3876
    1.2.840.113556.1.4.319 - LDAP Simple Paged Results - Control - RFC2696

[lines omitted for brevity]

    1.3.6.1.4.1.4203.1.5.4 - Language Tag Options - Feature - RFC3866
    1.3.6.1.4.1.4203.1.5.5 - language Range Options - Feature - RFC3866
  Schema entry: cn=Subschema
Other:
  objectClass:
    top
    OpenLDAProotDSE
```

(continues on next page)

(continued from previous page)

```
  structuralObjectClass:
OpenLDAProotDSE
  configContext:
    cn=config
  monitorContext:
    cn=Monitor
  entryDN:
```

4.1.2 Get information about an objectClass

No one remembers every possible attribute for every objectClass off the top of their head. For information about what a particular attributes an objectClass supports or requires, you can use `class_info`. `class_info` will display all required and optional attributes for an objectClass, as well as all superior objectClasses it inherits attributes from. (You can specifically look up the details for only the current objectClass with the `-n`/`--no-superior` option).

```
ezldap class_info inetOrgPerson
```

```
Object class: 2.16.840.1.113730.3.2.2
  Short name: inetOrgPerson
  Description: RFC2798: Internet Organizational Person
  Type: Structural
  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

[more classes that inetOrgPerson is derived from follow...]
```

4.1.3 Search using LDAP filters

You can query an LDAP directory using `search`. This will use the same syntax as `ldapsearch`. For convenience, single filters do not need to be wrapped in parentheses (for example, `(objectClass=*)` can be represented with `objectClass=*`). More complex queries should be wrapped in parentheses and quotes: `(&(cn=someuser)(objectClass=posixAccount))` should be represented as `'(&(cn=someuser)(objectClass=posixAccount))'`.

```
ezldap search objectClass=organizationalUnit
```

```
dn: ou=Group,dc=ezldap,dc=io
objectClass: organizationalUnit
ou: Group

dn: ou=People,dc=ezldap,dc=io
objectClass: organizationalUnit
ou: People

dn: ou=Hosts,dc=ezldap,dc=io
objectClass: organizationalUnit
ou: Hosts
```

4.1.4 Search for entries by DN

This function finds any DNs in a directory tree matching a keyword. (Might not work for huge directories yet due to paging limits.)

```
ezldap search_dn People
```

```
ou=People,dc=ezldap,dc=io
```

4.2 Add entries

ezldap supports adding entries to a directory using a set of configurable LDIF templates in `~/.ezldap`. Let's go through some example use cases.

4.2.1 Add a group

```
ezldap add_group demo
```

```
Success!
```

Verify the group has been created using `ezldap search` (you can also use `ldapsearch`, it won't hurt my feelings...)

```
ezldap search cn=demo
```

```
dn: cn=demo,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
cn: demo
gidNumber: 10000
```

4.2.2 Add a group using an alternate LDIF template

Chances are, the default LDIFs provided in this package won't match your organization's needs. No problem - ezldap works off of templates you can customize to your needs. Let's create a copy of the default `add_group.ldif` and use that instead:

```
cp ~/.ezldap/add_group.ldif custom_group.ldif
vim custom_group.ldif
```

Perhaps we want our new group to be an `extensibleObject` in addition to `top` and `posixGroup`. Our `custom_group.ldif` might look like this instead:

```
dn: cn=$groupname,$groupdn
objectClass: top
objectClass: posixGroup
objectClass: extensibleObject
cn: $groupname
gidNumber: $gid
```

To use this custom LDIF instead of the default, we can specify the path to our custom LDIF as a command-line option (if we wanted to use this as the default, we could have edited `~/.ezldap/add_group.ldif` instead):

```
ezldap add_group --ldif custom_group.ldif our-custom-group
```

```
Success!
```

Let's check our work and make sure our group was created:

```
ezldap search objectClass=extensibleObject
```

```
dn: cn=our-custom-group,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
objectClass: extensibleObject
cn: our-custom-group
gidNumber: 10001
```

4.2.3 Add a user

Creating a user is similar to creating a group. The only thing to remember here is that if we do not specify a group to add a user to, one will be created with the same name as that user:

```
ezldap add_user jeff
```

```
Creating LDAP group jeff... Success!
Creating user jeff... Success!
Adding jeff to LDAP group jeff... Success!
Password: 4NEy5uTs47
```

Checking our work:

```
ezldap search cn=jeff
```

```
dn: cn=jeff,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
cn: jeff
gidNumber: 10002
memberUid: jeff

dn: uid=jeff,ou=People,dc=ezldap,dc=io
objectClass: top
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
cn: jeff
sn: jeff
loginShell: /bin/bash
uidNumber: 10000
gidNumber: 10002
gecos: jeff
shadowMax: 180
shadowWarning: 7
```

(continues on next page)

(continued from previous page)

```
homeDirectory: /home/jeff
uid: jeff
```

As with `add_group`, all of the steps (adding a user, adding a group, adding the user to that group) let you customize which LDIFs get used.

4.2.4 Add a user to a group

Let's add `jeff` to our `demo` group from earlier.

```
ezldap add_to_group jeff demo
```

```
Success!
```

Check our work:

```
ezldap search cn=demo
```

```
dn: cn=demo,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
cn: demo
gidNumber: 10000
memberUid: jeff
```

It looks like `jeff` was successfully added.

4.3 Modify an entry

ezldap provides a modify command that can modify any attribute of an entry (add, replace, delete). What happens if we want to change the gidnumber of the `demo` group?

4.3.1 modify replace

```
ezldap modify cn=demo,ou=Group,dc=ezldap,dc=io replace gidNumber 12345
```

```
Success!
```

If we search for the demo group again, it should now reflect the new gidNumber:

```
ezldap search cn=demo
```

```
dn: cn=demo,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
cn: demo
memberUid: jeff
gidNumber: 12345
```

4.3.2 modify delete

What if we want to delete "jeff" as a member?

```
ezldap modify cn=demo,ou=Group,dc=ezldap,dc=io delete memberUid jeff
```

```
Success!
```

Result:

```
dn: cn=demo,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
cn: demo
gidNumber: 12345
```

4.3.3 modify add

Let's restore jeff as a member and add that user back into the group:

```
ezldap modify cn=demo,ou=Group,dc=ezldap,dc=io add memberUid jeff
```

Result:

```
dn: cn=demo,ou=Group,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
cn: demo
gidNumber: 12345
memberUid: jeff
```

4.3.4 Renaming / moving objects

The `modify_dn` operation lets you rename and/or move objects around in a directory. For convenience, the modify_dn provided by the `ezldap` command lets you both move and rename an entry in one go. To rename the `demo` group to `new-name` and move it into the `ou=People` container:

```
ezldap modify_dn cn=demo,ou=Group,dc=ezldap,dc=io cn=new-name,ou=People,dc=ezldap,
↪dc=io
```

```
ezldap search cn=new-name
```

Result:

```
dn: cn=new-name,ou=People,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
gidNumber: 12345
memberUid: jeff
cn: new-name
```

4.4 Miscellaneous operations

4.4.1 Delete an object

Maybe we realized that putting a group in the `ou=People` organizationalUnit was a bad idea. Maybe we just didn't want the `new-name` group anymore. Let's delete it. Note - since this is an inherently risky operation, you'll be prompted for confirmation before deleting anything (unless you use the `-f`/`--force` option).

```
ezldap delete cn=new-name,ou=People,dc=ezldap,dc=io
```

```
dn: cn=new-name,ou=People,dc=ezldap,dc=io
objectClass: top
objectClass: posixGroup
gidNumber: 12345
memberUid: jeff
cn: new-name

Delete object? (y/N) y
Success!
```

4.4.2 Change a user's password

Users frequently forget passwords. Though hopefully you won't have to reset passwords manually for users every time, there's a convenience function to speed things up: `change_pw`. In this case, the `-s` option lets us specify a new password. To simply randomize it, leave this option out.

```
ezldap change_pw -s jeff
```

```
New password for jeff:
Confirm password:
Success!
```

4.4.3 Check a user's password

Are you sure you typed that right? Absolutely sure? Let's check with `check_pw`:

```
ezldap check_pw jeff
```

```
Enter password to verify...
Password:
Passwords match!
```

4.5 Other commands / help

I've covered a few of the more common commands here. For more information on commands, refer to the `ezldap` client's command-line documentation (just add either the `-h` or `--help` options to bring up detailed help for each command).

```
ezldap --help
```

```
usage: ezldap [-h] [-v]  ...

ezldap CLI - Perform various options on an LDAP directory.

optional arguments:
 -h, --help     show this help message and exit
 -v, --version  show program's version number and exit

Valid commands:

  config        Configure ezldap (configs are stored in ~/.ezldap/).
  search        Search for entities by LDAP filter.
  search_dn     Search for and print DNs in a directory that match a keyword.
  add_user      Add a user.
  add_group     Add a group.
  add_to_group  Add a user to a group.
  add_host      Add a host.
  add_ldif      Add a generic LDIF template to a directory.
  modify        Add, replace, or delete an attribute from an entity.
  modify_ldif   Modify an entry using an LDIF template.
  modify_dn     Rename the DN of and/or move an entry.
  delete        Delete an entry from an LDAP directory.
  change_home   Change a user's home directory.
  change_shell  Change a user's default shell.
  change_pw     Change or reset a user's password.
  check_pw      Check a user's password.
  bind_info     Print info about ezldap's connection to your server.
  server_info   Print information about the LDAP server you are using.
  class_info    Print information about a specific LDAP objectClass.
```

For help on a given command:

```
ezldap modify --help
```

```
usage: ezldap modify [-h] dn {add,replace,delete} attribute value [replace_with]

Add, replace, or delete an attribute from an entity.

positional arguments:
  dn                    Distinguished Name (DN) of object to modify.
  {add,replace,delete}  Type of operation to perform. Can be one of: add,
                        replace, delete.
  attribute             Attribute to modify.
  value                 Value to add, replace, or delete. When performing a
                        delete operation, passing "-" will delete all values
                        for that attribute.
  replace_with          Value to replace an attribute with when performing a
                        replace operation.

optional arguments:
  -h, --help            show this help message and exit
```

4.6 A note on errors

If you run into an error, the `ezldap` client will immediately exit and print the reason for the error. Operations do not get performed half-way and leave things in a broken state. That said, I provide no guarantees or warranty of any kind while using this package. If you want to check that things are working correctly, run the tests! (You can also spin up a custom LDAP instance using a tool like Docker an test against that, you can use this package's Dockerfile as a reference to build your own test instances.)

Example error:

```
ezldap class_info sldfjsldjfl
```

```
objectClass "sldfjsldjfl" not found.
```

CHAPTER 5

Python recipes

This is a "recipe book" of things that can be done using the ezldap Python API. These examples all assume that you've imported the *ezldap* package like so:

```
import ezldap
```

You *must* run `ezldap config` on the command-line before the package will work.

5.1 Bind to a directory

5.1.1 Anonymous bind

```
connection = ezldap.Connection('ldap:///')
```

5.1.2 Bind using your credentials in `~/.ezldap`

```
connection = ezldap.auto_bind()
```

5.1.3 Bind manually

```
connection = ezldap.Connection('ldap:///', user='cn=someuser,dc=example,dc=com',
    password='password')
```

5.1.4 Unbind from a directory

```
connection.unbind()
```

5.1.5 Recommended workflow

Though it is possible to define a connection and later unbind. It is often easier to just use the `with` keyword, that will unbind for you.

```
with ezldap.auto_bind() as con:
    # do something with the "con" connection
```

(More documentation is on its way here, taking a break for now. . .)

CHAPTER 6

Python API

This is the page for ezldap's Python API documentation. For general-purpose recipes and use cases, check out the Python recipes.

6.1 Connection

ezldap.**ping**(*uri*)
Returns true if an LDAP server is responding at a given URI by attempting an anonymous bind.

ezldap.**supports_starttls**(*uri*)
Determine if the server actually supports StartTLS (both the server software itself supports it, and the server instance itself has been configured with SSL support).

ezldap.**auto_bind**(*conf=None*, *server_info=True*)
Automatically detects LDAP config values and returns a directory binding.

ezldap.**dn_address**(*dn*)
Get the "."-delmited address for a DN (typically a directory naming context/ base dn). If the directory naming context was dc=ezldap,dc=io, then the address would be "ezldap.io". Typically used when generating fully-qualified hostnames for new hosts (for instance, "hostname.ezldap.io"). However, this function will create addresses for any DN, if so desired.

ezldap.**clean_uri**(*uri*)
ldap3 really struggles with URIs ending in a slash. This function cleans common LDAP URI formats to something the underlying ldap3 API can understand.

class ezldap.**Connection**(*host*, *user=None*, *password=None*, *conf=None*, *authentication='SIMPLE'*, *server_info=True*, *client_strategy='SYNC'*, *sasl_mechanism=None*, *sasl_credentials=None*)
An object-oriented wrapper around an LDAP connection. To automatically create a binding use ezldap.auto_bind() instead. If either user or password are omitted, the bind is anonymous. conf is a dictionary with any placeholders or key/value combinations that you wish to be passed to ldif templates or details like user/group OUs. When used with the "with" keyword, the connection will automatically call Connection.unbind() (and unbind from the directory) when done.

__init__(*host*, *user=None*, *password=None*, *conf=None*, *authentication='SIMPLE'*, *server_info=True*, *client_strategy='SYNC'*, *sasl_mechanism=None*, *sasl_credentials=None*)

Parameters

- **host** – An LDAP server URI (eg. ldaps://someserver:636)
- **user** – Bind user. If None, bind will be anonymous.
- **password** – Bind password. If None, the bind will be anonymous.
- **conf** – A dict of configuration falues, such as those generated by ezldap.config().
- **client_strategy** – Communication strategy used by the client (defaults to SYNC).
- **authentication** – Type of authentication to use, by default ldap3.SIMPLE
- **sasl_mechanism** – The SASL mechanism to use for AUTH_SASL authentication. Available mechanisms are EXTERNAL, DIGEST-MD5 (deprecated by RFCs because insecure) and GSSAPI.
- **sasl_credentials** – An object specific to the SASL mechanism chosen. Refer to the documentation for each SASL mechanism supported.
- **server_info** – Whether to fetch information about the server like schema and supported controls. Setting this to False will significantly increase speed of the bind.

Returns Returns a directory binding used to perform operations on a directory.

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

add_group(*groupname*, *ldif_path='~/.ezldap/add_group.ldif'*, ***kwargs*)
Adds a group from an LDIF template.

add_host(*hostname*, *ip_address*, *ldif_path='~/.ezldap/add_host.ldif'*, ***kwargs*)
Add a host to a directory. Hostname is the short hostname (hostname -s), to add. If specifying the fully qualified hostname is desired (or the fully qualified hostname does not match the directory suffix), specify the fully-qualified hostname as "hostname_fq".

add_to_group(*username*, *groupname*, *ldif_path='~/.ezldap/add_to_group.ldif'*, ***kwargs*)
Adds a user to a group. The user and group in question must already exist.

add_user(*username*, *groupname*, *password*, *ldif_path='~/.ezldap/add_user.ldif'*, ***kwargs*)
Adds a user. Does not create or modify groups. "groupname" may be None if "gid" is specified.

base_dn()
Detect the base DN/naming context from an LDAP connection.

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.

exists(*dn*)
Returns true if a given DN exists in an LDAP directory.

extended(*request_name*, *request_value=None*, *controls=None*, *no_encode=None*)
Performs an extended operation

get_group(*group*, *basedn=None*, *index='cn'*)
Return a given group. Searches entire directory if no base search dn given.

get_host(*host*, *basedn=None*, *index='cn'*)
Return a given host. Searches entire directory if no base search dn given.

get_user(*user*, *basedn=None*, *index='uid'*)
Return given user as a dict or None if none is found. Searches entire directory if no base search dn given.

ldif_add(*ldif*)
Perform an add operation using an LDIF object.

ldif_modify(*ldif*)
Perform an LDIF modify operation from an LDIF object.

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_add(*dn*, *attrib*, *value*)
Add a single attribute to an object.

modify_delete(*dn*, *attrib*, *value=None*)
Delete a single attribute from an object. If value is None, deletes all attributes of that name.

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.

modify_replace(*dn*, *attrib*, *value*, *replace_with=None*)
Change a single attribute on an object.

next_gidn(*search_filter='(objectClass=posixGroup)'*, *search_base=None*, *gid_start=10000*, *gid_attribute='gidNumber'*)
Determine the next available gid number in a directory tree.

next_uidn(*search_filter='(objectClass=posixAccount)'*, *search_base=None*, *uid_start=10000*, *uid_attribute='uidNumber'*)
Determine the next available uid number in a directory tree.

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 noRFC2696 with the specified size

- If paged is 0 and cookie is present the search is abandoned on server 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
- 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

search_df(*search_filter='(objectClass=*)'*, *attributes='*'*, *search_base=None*, ***kwargs*)
A convenience function to search an LDAP directory and return a Pandas DataFrame. Very useful for analyzing the contents of your directory, computing stats, etc. Requires the pandas package to be installed.

search_list(*search_filter='(objectClass=*)'*, *attributes='*'*, *search_base=None*, ***kwargs*)
A wrapper around search() with better defaults and output format. A list of dictionaries will be returned, with one dict per output object. If search_base is None, the directory base DN will be used.

Parameters

- **search_filter** – An LDAP search filter.
- **attributes** – Attributes to return. If not specified, all defaults will be returned. None will return no attributes.
- **search_base** – Level of directory to begin search at, for example ou=People.

Returns A list of dicts, one per entry returned.

search_list_t(*search_filter='(objectClass=*)'*, *attributes='*'*, *search_base=None*, *unpack_lists=True*, *unpack_delimiter='|'*, ***kwargs*)
A utility function that returns the transposed result of search_list() (a dict of lists, with one list per attribute.) This is very useful for tasks like retrieving all uidNumbers currently assigned or emails used by users. The DN of each entry is always output.

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

who_am_i()
Return the DN of the user you have currently connected as.

6.2 LDIF parser and utilities

ezldap.**ldif_read**(*path*, *replacements=None*)
Read an LDIF file into a list of dicts appropriate for use with ezldap.

Parameters

- **path** – Path of an LDIF file to read.

- **replacements** – A dictionary of replacement values to replace $placeholders in the LDIF template.

ezldap.**ldif_write**(*entries*, *path*)
Write self.entries as LDIF file.

Parameters

- **entries** – A list of dicts, such as that returned by Connection.search_list()
- **path** – File to write.

ezldap.**ldif_print**(*entries*)
Print an LDIF entry to stdout.

Parameters **entries** – A list of dicts, such as that returned by Connection.search_list().

ezldap.**template**(*path*, *replacements=None*)
Read a file and substitute replacment entries for placeholders designated by $placeholder_name. If replacements is None, it simply opens and reads a file into a string.

6.3 Password utilities

ezldap.**random_passwd**(*length=10*, *ambiguous_chars=False*)
Generate a readable, random password with no ambiguous characters (unless you set that option to true, of course).

ezldap.**ssha_passwd**(*str_val*)
Hash and salt a string using SHA1 algorithm and format for use with LDAP.

ezldap.**ssha_check**(*ssha_val*, *str_val*)
Check that a password decodes to the correct password.

ezldap.**ssha**(*val*, *salt*)
Generate an SSHA hash.

CHAPTER 7

ldap3 and ezldap compatibility

At its heart, ezldap is is an extension to the ldap3 package, specifically the ldap3.Connection class. The ezldap Connection class extends ldap3.Connection with additional convenience methods and streamlines working with an active LDAP connection. ezldap is 100% compatible with the ldap3 API with a few notable exceptions:

7.1 Differences in search

The `search_list()`, `search_list_t()`, and `search_df()` methods provide differently formatted output than ldap3's `search()` (which is still available for use). `search_list()` treats returned entries as a list of dicts, one for each entry, with each attribute/objectClass/DN as a list in the dict. ldap3's `search()` returns a list of nested dictionaries, with DNs, objectClasses, and attributes handled separately. This simplification was made for convenience - ldap3's deep nesting of search results and inconsistent output format was inconvenient to parse and prone to error.

7.2 LDIF parser

The ezldap LDIF parsing and writing functions work in the same manner as the search functionality and produce identical output. As such, the LDIF writing tools and ldapadd/ldapchange functionality will only work when used with the corresponding ezldap functions.

CHAPTER 8

A note on security

ezldap tries to do things the right way. It will attempt to force a StartTLS operation before binding in all cases, and connecting to an `ldaps://` URI or over port 636 will connect using SSL. Encryption is preferred by default. A cleartext bind will only be performed if the server supports neither StartTLS or SSL (and it will warn you when it does so!).

All of that said, one of the configuration options is to specify your bind password as part of the config. *I highly recommend leaving this option blank.* This would store your bind password in plaintext in `~/.ezldap/config.yml`. Don't do it! (The option is there purely for convenience while testing and maybe if you wanted to add a huge swath of users from the command line.)

Instead of specifying your password using `ezldap config`, just leave the bind password field blank to be prompted for your password every time you perform a bind using the bind DN (typically the directory manager). If you've already specified a password and want to remove it, just delete the corresponding value for `bindpw` in `~/.ezldap/config.yml`.

Example:

```
# assuming "bindpw" is not specified in ~/.ezldap/config.yml
ezldap add_host compute-node 10.100.1.123
```

```
Enter bind DN password...
Success!
```

Operations that can be performed anonymously (using an anonymous bind without credentials) are preferred by `ezldap` whenever possible. Generally `ezldap` will only prompt you for a bind password if it needs it.

CHAPTER 9

Supported servers

Though the underlying ldap3 API that ezldap is built on supports virtually all vendors, ezldap is currently only tested against OpenLDAP. Most ezldap operations will likely work against all LDAP server implementations, though this has not been tested!

What happens if an operation fails? In all likelihood, nothing - the server will refuse to perform the operation and ezldap will raise an error. That said, there are bound to be bugs until ezldap has test cases targeting other LDAP server implementations.

If you encounter a bug, please raise an issue on the issue tracker on Github.

The current list of LDAP servers I intend to add compatibility for is as follows:

- OpenLDAP (done)
- 389 Directory Server / FreeIPA
- Active Directory

Index

Symbols

A

B

C

D

E

G

L

M

N

P

R

S

T

U

W