
laurelin Documentation

Release 1.5.3

Alex Shafer

Nov 17, 2018

Contents

1	User Docs	3
1.1	Features Overview	3
1.2	Missing/incomplete features	4
1.3	Walkthrough	4
1.3.1	Navigating	4
1.3.2	Getting Started	5
1.3.3	LDAP Methods Intro	5
1.3.4	LDAPObject Methods Intro	6
1.4	Relative Searching	7
1.5	Attributes Dictionaries	7
1.6	Modify Operations	8
1.6.1	Raw modify methods	8
1.6.2	Strict modification and higher-level modify functions	8
1.7	Global Defaults, LDAP instance attributes, and LDAP constructor arguments	9
1.8	Basic usage examples	10
1.8.1	1. Connect to local LDAP instance and iterate all objects	10
2	Simple Search Filters	11
3	Config Files	13
3.1	Intro	13
3.2	Global Section	14
3.3	Extensions Section	14
3.4	Connection Section	14
3.5	Objects Section	14
3.6	Global vs. Connection	15
3.7	Load Order	15
3.8	Using Dicts Directly	15
4	Built-In Extensions	17
4.1	Description Attributes	17
4.2	NIS Netgroups	18
4.3	Paged Results	19
5	Creating Extensions	21
5.1	Extension Activation	21
5.2	Binding New Methods	22

5.3	LDAP Extensions	22
5.4	Controls	23
5.5	Schema	24
5.5.1	Object Classes and Attribute Types	24
5.5.2	Matching Rules	24
5.5.3	Syntax Rules	25
5.5.4	SchemaValidator	25
5.6	Validators	26
5.7	Packaging	26
6	Controls	27
6.1	Using Controls	27
6.2	Defining Controls	28
7	Changelog	31
7.1	1.5.3	31
7.2	1.5.2	31
7.3	1.5.0	31
7.4	1.4.1	31
7.5	1.4.0	32
7.6	1.3.1	32
7.7	1.3.0	32
7.8	1.2.0	32
7.9	1.1.0	33
8	Reference	35
8.1	laurelin.ldap package	35
8.1.1	Submodules	35
8.1.2	laurelin.ldap.attributetype module	35
8.1.3	laurelin.ldap.attrsdict module	38
8.1.4	laurelin.ldap.attrvaluelist module	38
8.1.5	laurelin.ldap.base module	39
8.1.6	laurelin.ldap.config module	53
8.1.7	laurelin.ldap.constants module	55
8.1.8	laurelin.ldap.exceptions module	56
8.1.9	laurelin.ldap.extensible module	58
8.1.10	laurelin.ldap.filter module	58
8.1.11	laurelin.ldap.ldapobject module	59
8.1.12	laurelin.ldap.modify module	66
8.1.13	laurelin.ldap.net module	66
8.1.14	laurelin.ldap.objectclass module	68
8.1.15	laurelin.ldap.rules module	69
8.1.16	laurelin.ldap.schema module	71
8.1.17	laurelin.ldap.validation module	80
8.1.18	Module contents	80
9	Testing Setup	101
9.1	System	101
9.2	SASL	101
9.2.1	SASL config ldif	101
9.2.2	Adding sasl user password with	102
9.2.3	SASL auth control test case	102
9.3	LDAPS/StartTLS	102
10	Indices and tables	103

Laurelin is a pure-Python ORM-esque LDAP client. Check out the [user docs](#) to get started. View the source on [GitHub](#).

- *Features Overview*
- *Missing/incomplete features*
- *Walkthrough*
 - *Navigating*
 - *Getting Started*
 - *LDAP Methods Intro*
 - *LDAPObject Methods Intro*
- *Relative Searching*
- *Attributes Dictionaries*
- *Modify Operations*
 - *Raw modify methods*
 - *Strict modification and higher-level modify functions*
- *Global Defaults, LDAP instance attributes, and LDAP constructor arguments*
- *Basic usage examples*
 - *1. Connect to local LDAP instance and iterate all objects*

1.1 Features Overview

- Fully compliant with RFC 4510 and its children.
- Pure Python codebase, meaning that it can be used with Python implementations other than CPython.

- Tested against CPython 2.7, 3.3, 3.4, 3.5, 3.6, PyPy, and PyPy3.
- Simplified filter syntax (optional, standard filter syntax is fully supported and used by default)
- Pythonic attributes input and presentation. It's just a dictionary.
- Exceedingly easy relative searching. All objects have a suite of search methods which will automatically pass the object's DN as the search base. In many cases, you won't have to pass *any* arguments to search methods.
- Similarly, all objects have a suite of modify methods which allow you to change attributes on already-queried objects without having to pass their DN again.
- Intelligent modification will never send existing attribute values to the server, nor will it request deletion of attribute values that do not exist. This prevents many unnecessary server errors. Laurelin will go as far as to query the object for you before modifying it to ensure you don't see pointless errors (if you want it to).
- Custom validation. You can define validators which check new objects and modify operations for correctness before sending them to the server. Since you control this code, this can be anything from a simple regex check against a particular attribute value, to a complex approval queue mechanism.
- Highly extensible. New methods can easily and safely be bound to base classes.
- Seamless integration of controls. Once defined, these are just new keyword arguments on particular methods, and additional attributes on the response object.
- Includes Python implementations of standard schema elements. This conveys many benefits:
 - Allows changes to be validated *before* sending the server
 - Allows matching rules to be used to compare attribute values locally. Many attribute types are case-insensitive and have other rules meaning that the standard Python `==` or `in` operators won't tell you what you want to know. Laurelin makes them work according to these rules.

1.2 Missing/incomplete features

Some lesser-used features of the LDAP protocol have not yet been implemented or are incomplete. Check the [GitHub issues](#) to see if your use case is affected. Please add a comment if so, or open a new issue if you spot anything else. PRs are always welcome.

1.3 Walkthrough

Note: I'm assuming that if you're here, you're already pretty familiar with LDAP fundamentals. If you don't know how to write a search filter, you may want to do some more reading on LDAP before continuing.

1.3.1 Navigating

Just about everything you need for routine user tasks is available in the `laurelin.ldap` package. `laurelin.ldap.exceptions` contains all exception definitions which you may need to import to catch, but even some common ones are included in `laurelin.ldap`. Beyond that, you should not need to get into the sub-modules unless you are defining controls, extensions, schema, or validators.

Built-In Extensions are stored in the `laurelin.extensions` package.

1.3.2 Getting Started

The first thing you should typically do after importing is configure logging and/or warnings. There is a lot of useful information available at all log levels:

```
from laurelin.ldap import LDAP

LDAP.enable_logging()
# Enables all log output on stderr
# It also accepts an optional log level argument, e.g. LDAP.enable_logging(logging.
↳ERROR)
# The function also returns the handler it creates for optional further manual_
↳handling

import logging

logger = logging.getLogger('laurelin.ldap')
# Manually configure the logger and handlers here using the standard logging module
# Submodules use the logger matching their name, below laurelin.ldap

LDAP.log_warnings()
# emit all LDAP warnings as WARN-level log messages on the laurelin.ldap logger
# all other warnings will take the default action

LDAP.disable_warnings()
# do not emit any LDAP warnings
# all other warnings will take the default action
```

You can then initialize a connection to an LDAP server. Pass a URI string to the `LDAP` constructor:

```
with LDAP('ldap://dir.example.org:389') as ldap:
    # do stuff...

# Its also possible, but not recommended, to not use the context manager:
ldap = LDAP('ldap://dir.example.org:389')
```

This will open a connection and query the server to find the “base DN” or DN suffix. An empty `LDAPObject` will be created with the base DN and stored as the `base` attribute on the `LDAP` instance. More on this later. For now we will briefly cover the basic LDAP interface which may seem somewhat familiar if you have used the standard python-ldap client before.

1.3.3 LDAP Methods Intro

`LDAP.search()` sends a search request and returns an iterable over instances of `LDAPObject`. Basic arguments are described here (listed in order):

- `base_dn` - the absolute DN to start the search from
- `scope` - One of:
 - `Scope.BASE` - only search `base_dn` itself
 - `Scope.ONE` - search `base_dn` and its immediate children
 - `Scope.SUB` - search `base_dn` and all of its descendents (default)
- `filter` - standard LDAP filter string
- `attrs` - a list of attributes to return for each object

Use `LDAP.get()` if you just need to get a single object by its DN. Also accepts an optional list of attributes.

`LDAP.add()` adds a new object, and returns the corresponding `LDAPObject`, just pass the full, absolute DN and an *attributes dict*

`LDAP.delete()` deletes an entire object. Just pass the full, absolute DN of the object to delete.

The following methods are preferred for modification, however raw *modify methods* are also provided.

All accept the absolute DN of the object to modify, and an *attributes dictionary*.

`LDAP.add_attrs()` adds new attributes.

`LDAP.delete_attrs()` deletes attribute values. Pass an empty values list in the attributes dictionary to delete all values for an attribute.

`LDAP.replace_attrs()` replaces all values for the given attributes with the values passed in the attributes dictionary. Attributes that are not mentioned are not touched. Passing an empty list removes all values.

For `LDAP.delete_attrs()` and `LDAP.replace_attrs()` you can specify the constant `LDAP.DELETE_ALL` in place of an empty attribute value list to remove all values for the attribute. For example:

```
ldap.replace_attrs('cn=foo,dc=example,dc=org', {'someAttribute': LDAP.DELETE_ALL})
```

If you wish to require the use of the constant instead of an empty list, pass `error_empty_list=True` to the `LDAP` constructor. You can also pass `ignore_empty_list=True` to silently prevent these from being sent to the server (this will be the default behavior in a future release).

1.3.4 LDAPObject Methods Intro

Great, right? But specifying absolute DNs all the time is no fun. Enter `LDAPObject`, and keep in mind the base attribute mentioned earlier.

`LDAPObject` inherits from `AttrsDict` to present attributes. This interface is documented [here](#).

`LDAPObject` defines methods corresponding to all of the `LDAP` methods, but pass the object's dn automatically, or only require the RDN prefix, with the object's dn automatically appended to obtain the absolute DN.

`LDAPObject.search()` accepts all the same arguments as `LDAP.search()` except `base_dn` and `scope`. The object's own DN is always used for `base_dn`, and the `relative_search_scope` is always used as the `scope`.

`LDAPObject.find()` is more or less a better `LDAPObject.get_child()`. It looks at the object's `relative_search_scope` property to determine the most efficient way to find a single object below this one. It will either do a `BASE` search if `relative_search_scope=Scope.ONE` or a `SUBTREE` search if `relative_search_scope=Scope.SUB`. It is an error to use this method if `relative_search_scope=Scope.BASE`.

`LDAPObject.get_child()` is analagous to `LDAP.get()` but it only needs the RDN, appending the object's own DN as mentioned earlier. (Note that `LDAPObject.get()` inherits from the native `dict.get()`)

`LDAPObject.add_child()` is analagous to `LDAP.add()` again accepting an RDN in place of a full absolute DN.

Use `LDAPObject.get_attr()` like `dict.get()` except an empty list will always be returned as default if the attribute is not defined.

`LDAPObject`'s modify methods update the server first, then update the local attributes dictionary to match if successful. `LDAPObject.add_attrs()`, `LDAPObject.delete_attrs()`, and `LDAPObject.replace_attrs()` require only a new attributes dictionary as an argument, of the same format as for the matching `LDAP` methods.

`LDAPObject` Examples:

```

people = ldap.base.get_child('ou=people')

print(people['objectClass'])
# ['top', 'organizationalUnit']

people.add_attrs({'description':['Contains all users']})

# list all users
for user in people.search(filter='(objectClass=posixAccount)'):
    print(user['uid'][0])

```

1.4 Relative Searching

All objects have `LDAPObject.search()` and `LDAPObject.find()` methods which utilize the `relative_search_scope` attribute of the object. `relative_search_scope` can be passed as a keyword to any method that creates new objects, including `LDAP.obj()`, `LDAP.get()`, `LDAP.search()`, `LDAP.add()`, `LDAPObject.obj()`, `LDAPObject.find()`, `LDAPObject.search()`, `LDAPObject.get_child()`, and `LDAPObject.add_child()`.

When you create an object from another `LDAPObject` and you *don't* specify the `relative_search_scope`, it is automatically inherited from the parent object. When you create an object from an `LDAP` method, it defaults to `Scope.SUB`.

The real win with this feature is when your tree is structured such that you can set this to `Scope.ONE` as this conveys significant performance benefits, especially when using `LDAPObject.find()`. This allows laurelin to construct the absolute DN of the child object and perform a highly efficient `BASE` search.

1.5 Attributes Dictionaries

This common interface is used both for input and output of LDAP attributes. In short: dict keys are attribute names, and dict values are a list of attribute values. For example:

```

{
    'objectClass': ['posixAccount', 'inetOrgPerson'],
    'uid': ['ashafer01'],
    'uidNumber': ['1000'],
    'gidNumber': ['100'],
    'cn': ['Alex Shafer'],
    'homeDirectory': ['/home/ashafer01'],
    'loginShell': ['/bin/zsh'],
    'mail': ['ashafer01@example.org'],
}

```

Note that there is an `AttrsDict` class defined - there is **no requirement** to create instances of this class to pass as arguments, though you are welcome to if you find the additional methods provided this class convenient, such as `AttrsDict.get_attr()`. Further, it overrides dict special methods to enforce type requirements and enable case-insensitive keys.

Also note that when passing an attributes dictionary to `LDAP.replace_attrs()` or `LDAP.delete_attrs()` it is legal to specify the constant `LDAP.DELETE_ALL` in place of a value list.

1.6 Modify Operations

1.6.1 Raw modify methods

`LDAP.modify()` and `LDAPObject.modify()` work similarly to the modify functions in `python-ldap`, which in turn very closely align with how modify operations are described at the protocol level. A list of `Mod` instances is required with 3 arguments:

1. One of the `Mod` constants which describe the operation to perform on an attribute:
 - `Mod.ADD` adds new attributes/values
 - `Mod.REPLACE` replaces all values for an attribute, creating new attributes if necessary
 - `Mod.DELETE` removes attributes/values.
2. The name of the attribute to modify. Each entry may only modify one attribute, but an unlimited number of entries may be specified in a single modify operation.
3. A list of attribute values to use with the modify operation or the constant `LDAP.DELETE_ALL`:
 - The list may be empty for `Mod.REPLACE` and `Mod.DELETE`, both of which will cause all values for the given attribute to be removed from the object. The list may not be empty for `Mod.ADD`. You can also specify the constant `LDAP.DELETE_ALL` in place of any empty list. If you wish to warn about empty lists or require the use of the constant, pass `warn_empty_list=True` or `error_empty_list=True` to the `LDAP` constructor. You can also pass `ignore_empty_list=True` to silently prevent these from being sent to the server (this will be the default behavior in a future release).
 - A non-empty list for `Mod.ADD` lists all new attribute values to add
 - A non-empty list for `Mod.DELETE` lists specific attribute values to remove
 - A non-empty list for `Mod.REPLACE` indicates ALL new values for the attribute - all others will be removed.

Example custom modify operation:

```
from laurelin.ldap.modify import Mod

ldap.modify('uid=ashafer01,ou=people,dc=example,dc=org', [
    Mod(Mod.ADD, 'mobile', ['+1 401 555 1234', '+1 403 555 4321']),
    Mod(Mod.ADD, 'homePhone', ['+1 404 555 6789']),
    Mod(Mod.REPLACE, 'homeDirectory', ['/export/home/ashafer01']),
])
```

Using an `LDAPObject` instead:

```
ldap.base.obj('uid=ashafer01,ou=people').modify([
    Mod(Mod.DELETE, 'mobile', ['+1 401 555 1234']),
    Mod(Mod.DELETE, 'homePhone', LDAP.DELETE_ALL), # delete all homePhone values
])
```

Again, an arbitrary number of `Mod` entries may be specified for each `modify` call.

1.6.2 Strict modification and higher-level modify functions

The higher-level modify functions (`add_attrs`, `delete_attrs`, and `replace_attrs`) all rely on the concept of *strict modification* - that is, to only send the modify operation, and to never perform an additional search. By default, strict modification is **disabled**, meaning that, if necessary, an extra search **will** be performed before sending a modify request.

You can enable strict modification by passing `strict_modify=True` to the `LDAP` constructor.

With strict modification disabled, the `LDAP` modify functions will engage a more intelligent modification strategy after performing the extra query: for `LDAP.add_attrs()`, no duplicate values are sent to the server to be added. Likewise for `LDAP.delete_attrs()`, deletion will not be requested for values that are not known to exist. This prevents many unnecessary failures, as ultimately the final semantic state of the object is unchanged with or without such failures. (Note that with `LDAP.replace_attrs()` no such failures are possible)

With the `LDAPObject` modify functions, the situation is slightly more complex. Regardless of the `strict_modify` setting, the more intelligent modify strategy will always be used, using at least any already-queried attribute data stored with the object (which could be complete data depending on how the object was originally obtained). If `strict_modify` is disabled, however, another search *may* still be performed to fill in any missing attributes that are mentioned in the passed attributes dict.

The raw modify functions on both `LDAP` and `LDAPObject` are unaffected by the `strict_modify` setting - they will always attempt the modify operation exactly as specified.

1.7 Global Defaults, LDAP instance attributes, and LDAP constructor arguments

All of the `LDAP` constructor arguments are set to `None` by default. In the constructor, any explicitly is `None` arguments are set to their associated global default. These are attributes of the `LDAP` class, have the same name as the argument, upper-cased, and with a `DEFAULT_` prefix (but the prefix wont be repeated).

For example, the `server` argument has global default `LDAP.DEFAULT_SERVER`, and `default_criticality` is `LDAP.DEFAULT_CRITICALITY`.

Most arguments also have an associated instance property. A complete table is below:

Global Default	LDAP instance attribute	LDAP constructor keyword
<code>LDAP.DEFAULT_SERVER</code>	<code>host_uri</code>	<code>server</code>
<code>LDAP.DEFAULT_BASE_DN</code>	<code>base_dn</code>	<code>base_dn</code>
<code>LDAP.DEFAULT_FILTER</code>	<code>none</code>	<code>none</code>
<code>LDAP.DEFAULT_DEREF_ALIASES</code>	<code>default_deref_aliases</code>	<code>deref_aliases</code>
<code>LDAP.DEFAULT_SEARCH_TIMEOUT</code>	<code>default_search_timeout</code>	<code>search_timeout</code>
<code>LDAP.DEFAULT_CONNECT_TIMEOUT</code>	<code>sock_params[0]</code>	<code>connect_timeout</code>
<code>LDAP.DEFAULT_STRICT_MODIFY</code>	<code>strict_modify</code>	<code>strict_modify</code>
<code>LDAP.DEFAULT_REUSE_CONNECTION</code>	<code>none</code>	<code>reuse_connection</code>
<code>LDAP.DEFAULT_SSL_VERIFY</code>	<code>ssl_verify</code>	<code>ssl_verify</code>
<code>LDAP.DEFAULT_SSL_CA_FILE</code>	<code>ssl_ca_file</code>	<code>ssl_ca_file</code>
<code>LDAP.DEFAULT_SSL_CA_PATH</code>	<code>ssl_ca_path</code>	<code>ssl_ca_path</code>
<code>LDAP.DEFAULT_SSL_CA_DATA</code>	<code>ssl_ca_data</code>	<code>ssl_ca_data</code>
<code>LDAP.DEFAULT_FETCH_RESULT_REFS</code>	<code>default_fetch_result_refs</code>	<code>fetch_result_refs</code>
<code>LDAP.DEFAULT_FOLLOW_REFERRALS</code>	<code>default_follow_referrals</code>	<code>follow_referrals</code>
<code>LDAP.DEFAULT_SASL_MECH</code>	<code>default_sasl_mech</code>	<code>default_sasl_mech</code>
<code>LDAP.DEFAULT_SASL_FATAL_DOWNGRADE_CHECK</code>	<code>default_sasl_fatal_downgrade_check</code>	<code>sasl_fatal_downgrade_check</code>
<code>LDAP.DEFAULT_CRITICALITY</code>	<code>default_criticality</code>	<code>default_criticality</code>
<code>LDAP.DEFAULT_VALIDATORS</code>	<code>validators</code>	<code>validators</code>
<code>LDAP.DEFAULT_WARN_EMPTY_LIST</code>	<code>warn_empty_list</code>	<code>warn_empty_list</code>
<code>LDAP.DEFAULT_ERROR_EMPTY_LIST</code>	<code>error_empty_list</code>	<code>error_empty_list</code>
<code>LDAP.DEFAULT_IGNORE_EMPTY_LIST</code>	<code>ignore_empty_list</code>	<code>ignore_empty_list</code>
<code>LDAP.DEFAULT_FILTER_SYNTAX</code>	<code>default_filter_syntax</code>	<code>filter_syntax</code>

The `LDAP` instance attributes beginning with `default_` are used as the defaults for corresponding arguments on other methods. `default_sasl_mech` is used with `LDAP.sasl_bind()`, `default_criticality` is the default criticality of all controls, the other `default_` attributes are used with `LDAP.search()`.

The `ssl_` prefixed instances attributes are used as the defaults for `LDAP.start_tls()`, as well as the socket configuration when connecting to an `ldaps://` socket.

1.8 Basic usage examples

1.8.1 1. Connect to local LDAP instance and iterate all objects

```
from laurelin.ldap import LDAP

with LDAP('ldapi:///') as ldap:
    ldap.sasl_bind()
    for obj in ldap.base.search():
        print(obj.format_ldif())
```

`LDAP.sasl_bind()` defaults to the EXTERNAL mechanism when an `ldapi: URI` is given, which uses the current user for authorization via the unix socket (Known as “autobind” with 389 Directory Server)

CHAPTER 2

Simple Search Filters

Laurelin provides an alternate syntax for search filters that is much simpler than the standard, RFC 4515-compliant, filter syntax. In short, it is a hybrid between SQL logic expressions and standard LDAP filter comparisons.

In the simplest case of a single comparison, the two syntaxes are identical:

Standard	Simple
(gidNumber=100)	(gidNumber=100)

But when it comes to expressing logic, the Laurelin simplified filter differs quite a bit:

Standard	Simple
(&(gidNumber<=1000) (! (memberUid=*)))	(gidNumber<=1000) AND NOT (memberUid=*)

Feel free to include parentheses in your simple filters if it helps clarify the logic:

Simple (without extra parens)	Simple (equivalent with extra parens)
(gidNumber<=1000) AND NOT (memberUid=*)	(gidNumber<=1000) AND (NOT (memberUid=*))

Some more equivalent standard and simple filters:

Standard	Simple
(&(abc=foo) ((def=bar) (ghi=ijkl)))	(abc=foo) AND ((def=bar) OR (ghi=ijkl))
((abc=foo) (&(def=bar) (ghi=ijkl)))	(abc=foo) OR (def=bar) AND (ghi=ijkl)
(&(abc=foo) ((def=bar) (ghi=ijkl)) (xyz=abc))	(abc=foo) AND ((def=bar) OR (ghi=ijkl)) AND (xyz=abc)

By default, Laurelin will interpret your filters with the **unified** filter syntax, meaning you can embed a full RFC 4515-compliant filter anywhere you see a simple comparison in the above examples. This includes as the only element in the filter, making this fully backwards compatible with RFC 4515 standard filters.

Currently available syntaxes are `FilterSyntax.STANDARD` to limit to RFC 4515, `FilterSyntax.SIMPLE` to limit to only simple comparisons within SQL-style logic, and the default `FilterSyntax.UNIFIED`.

If you wish to restrict the syntax, you can do one of the following:

1. Pass `filter_syntax=` to `LDAP.search()` or any other search method:

```
from laurelin.ldap import LDAP, FilterSyntax

with LDAP() as ldap:
    search = ldap.search('o=foo', filter='(abc=foo) AND (def=bar)', filter_
    ↪syntax=FilterSyntax.SIMPLE)
```

2. Pass `filter_syntax=` to the `LDAP` constructor:

```
from laurelin.ldap import LDAP, FilterSyntax

with LDAP(filter_syntax=FilterSyntax.SIMPLE) as ldap:
    search1 = ldap.search('o=foo', filter='(abc=foo) AND (def=bar)')
    search2 = ldap.search('o=bar', filter='(xyz=foo) OR (abc=bar)')
```

3. Set the global default `LDAP.DEFAULT_FILTER_SYNTAX` before instantiating any `LDAP` instances:

```
from laurelin.ldap import LDAP, FilterSyntax

LDAP.DEFAULT_FILTER_SYNTAX = FilterSyntax.STANDARD

with LDAP() as ldap:
    search = ldap.search('o=foo', filter='(&(abc=foo)(def=bar))')

with LDAP('ldap://localhost:10389') as ldap:
    search = ldap.search('o=bar', filter='(|(xyz=foo)(abc=bar))')
```

4. Do either of the two above using *Config Files*.

Note: How is this possible?

Doesn't the filter get sent to the server and parsed there like SQL? No! In LDAP, it is up to the client to parse the filter string into a set of objects that are encoded and sent to the server. If you've got any other ideas for alternate filter syntaxes, please submit a PR!

- *Intro*
- *Global Section*
- *Extensions Section*
- *Connection Section*
- *Objects Section*
- *Global vs. Connection*
- *Load Order*
- *Using Dicts Directly*

3.1 Intro

Laurelin config files may be YAML or JSON formatted out of the box. You can also supply your own custom decoding function to handle arbitrary formats. The important part is that the file contents decode to a dictionary. Below is an example YAML file:

```
global:
  SSL_CA_PATH: /etc/ldap/cacerts
  IGNORE_EMPTY_LIST: true
extensions:
  - laurelin.extensions.descattrs
  - laurelin.extensions.netgroups
connection:
  server: ldap://dir01.example.org
  start_tls: true
  simple_bind:
```

(continues on next page)

(continued from previous page)

```
username: testuser
passwd: testpassword
connect_timeout: 30
objects:
- rdn: ou=people
  tag: posix_user_base
- rdn: ou=groups
  tag: posix_group_base
- rdn: ou=netgroups
  tag: netgroup_base
```

You can load and apply such a file by using `config.load_file()`. If a connection section was specified, a new connection will be established and returned from the function.

3.2 Global Section

Each key in the global section must correspond to one of the `DEFAULT_` prefixed attributes on `LDAP`. As you can see in the example, the `DEFAULT_` prefix is optional. Not demonstrated by the example is that keys are case-insensitive (that is, they will be upper-cased for you).

3.3 Extensions Section

This is simply a list of extension module names which will get activated when the config file is loaded.

3.4 Connection Section

Keys here are *mostly* corresponding to `LDAP` constructor arguments, however there are a few special ones:

- `start_tls` A boolean option, if set to `true` will execute `LDAP.start_tls()` after opening the connection
- `simple_bind` A dictionary of parameters to pass to `LDAP.simple_bind()`
- `sasl_bind` A dictionary of parameters to pass to `LDAP.sasl_bind()`

Note that `simple_bind` and `sasl_bind` are both optional, and mutually exclude each other. In other words, it is an error to specify both of these keys.

Note that `start_tls` will always occur before any bind (if requested).

3.5 Objects Section

Note: You cannot specify `objects` without also specifying a connection

This is a list of dicts where keys correspond to `LDAP.obj()` or `LDAPObject.obj()` arguments. You *must* specify exactly one of `dn` or `rdn`. If `dn` is specified, this will be taken as the full, absolute DN of the object, and parameters will be passed to `LDAP.obj()`. If `rdn` is specified, this will be taken as the RDN relative to the connection's base object, or the base of the tree, and parameters will be passed to `LDAPObject.obj()`.

Also required for all objects is the `tag` key. This is how you will access created objects. For example, to access the first object in the config file example above:

```
ldap = config.load_file('/path/to/file.yaml')
posix_users = ldap.tag('posix_user_base')
```

Its important to note that the server is not queried when creating these objects, so they will not have any local attributes. If you require local attributes, you can all `LDAPObject.refresh()` on the object.

3.6 Global vs. Connection

As mentioned elsewhere in the docs, there is a global config parameter associated with every connection parameter, meaning in a config file you can define your connection parameters in either section. This *does not* have the exact same end functionality, though. In general you should prefer `connection` for the following reasons:

- The connection will not be created when the config file is loaded if you configure everything in `global`
- You cannot define `objects` without defining a `connection`
- You cannot specify `start_tls` or `bind` parameters globally

However there are cases where it may be desirable to specify everything as a global default. Taking this approach allows you to use the `LDAP` constructor with as few as zero arguments after loading the config. You can still bind as usual by calling `LDAP.simple_bind()` or `LDAP.sasl_bind()` on the connection. You can also manually create objects with `obj()` methods.

3.7 Load Order

Sections are loaded and applied in a specific order:

1. `global`
2. `extensions`
3. `connection`
4. `objects`

You can specify sections in whatever order is convenient in your file. They will *always* be used in the above order.

3.8 Using Dicts Directly

If you already have your configuration parameters in one or more dictionaries, you can apply them directly without going through the file interface. You can pass a dictionary of the same format as in a config file to `config.load_config_dict()`. Like `load_file()`, this will establish and return the new connection if one was defined.

You can also use the other `config` methods to apply dictionary configurations piecemeal. These process fragments of the larger config dictionary. Check the reference docs for details if you need to do this.

- *Description Attributes*
- *NIS Netgroups*
- *Paged Results*

4.1 Description Attributes

Support for structured description fields.

This implements the common pattern of storing arbitrary key=value data in description fields, but presents an attribute-like interface to access and change them.

Example:

```
from laurelin.ldap import LDAP
LDAP.activate_extension('laurelin.extensions.descattrs')

with LDAP() as ldap:
    result = ldap.base.get_child('cn=someObject')

    result.add_desc_attrs({'foo':['one', 'two']})
    print(result.format_ldif())
    # ...
    # description: foo=one
    # description: foo=two
    # ...

    attr_vals = result.desc_attrs().get_attr('foo')
    print(attr_vals)
    # ['one', 'two']
```

(continues on next page)

(continued from previous page)

```
result.replace_desc_attrs({'foo': ['one', 'two', 'three']})
result.delete_desc_attrs({'foo': ['two']})

attr_vals = result.desc_attrs().get_attr('foo')
print(attr_vals)
# ['one', 'three']

print(result.format_ldif())
# ...
# description: foo=one
# description: foo=three
# ...
```

class `laurelin.ldap.LDAPObject`

The following new methods get bound to `laurelin.ldap.LDAPObject` upon extension activation:

desc_attrs()

Query the description attribute if unknown and return an *AttrsDict* representing the data stored in the description.

Returns An *AttrsDict* representing the data stored in the description.

Return type *AttrsDict*

add_desc_attrs(attrs_dict)

Add new description attributes.

Parameters *attrs_dict* (*dict*(*str*, *list*[*str*]) or *AttrsDict*) – Dictionary of description attributes to add

Return type *None*

delete_desc_attrs(attrs_dict)

Delete description attributes.

Parameters *attrs_dict* (*dict*(*str*, *list*[*str*]) or *AttrsDict*) – Dictionary of description attributes to delete

Return type *None*

replace_desc_attrs(attrs_dict)

Replace description attributes.

Parameters *attrs_dict* (*dict*(*str*, *list*[*str*]) or *AttrsDict*) – Dictionary of description attributes to set

Return type *None*

4.2 NIS Netgroups

class `laurelin.ldap.LDAP`

The following new methods get bound to `laurelin.ldap.LDAP` upon extension activation:

class `laurelin.ldap.LDAPObject`

The following new methods get bound to `laurelin.ldap.LDAPObject` upon extension activation:

4.3 Paged Results

RFC 2696 Simple Paged Results Manipulation

This adds a control to support paging results. Use the control keyword `paged` with search methods. Returns a cookie on the `page_cookie` response attribute which can be found on the results handle after all paged results have been received. See example below.

Note: Do not use this extension to simply limit the total number of results. The search methods accept a `limit` keyword out of the box for this purpose.

Example usage:

```
from laurelin.ldap import LDAP
LDAP.activate_extension('laurelin.extensions.pagedresults')

with LDAP() as ldap:
    search = ldap.base.search(paged=10)
    page1_results = list(search)

    search = ldap.base.search(paged=(10, search.page_cookie))
    page2_results = list(search)

    # ...

    if not search.page_cookie:
        print('Got all pages')
```

Note: When getting pages in a loop, you may set the cookie value to an empty string on the first iteration, e.g.:

```
ldap.base.search(paged=(10, ''))
```

Creating Extensions

- *Extension Activation*
- *Binding New Methods*
- *LDAP Extensions*
- *Controls*
- *Schema*
 - *Object Classes and Attribute Types*
 - *Matching Rules*
 - *Syntax Rules*
 - *SchemaValidator*
- *Validators*
- *Packaging*

The most important thing to note about “extensions” is that they are not necessarily LDAP extensions. In *laurelin*, they are simply a module that binds additional methods to base classes (*LDAP*, or *LDAPObject*).

5.1 Extension Activation

The *LDAP.activate_extension()* method accepts a string containing the name of the module to import. After importing, an *activate_extension()* function will be called on the module itself if defined. Any setup can be done in this function, including calls to *EXTEND()* (see below).

5.2 Binding New Methods

In order to ensure all extensions play nicely together, **do not** bind methods to these yourself. Each extensible class has a classmethod called `EXTEND` which accepts a list of methods (or any callable object) to bind. If you need to bind the method as a different name, override it's `__name__` attribute before calling `EXTEND`.

No method may ever be overwritten, built-in or otherwise. If you want to modify the behavior of existing methods, you should create a subclass in your extension module and instruct your users to import this instead.

Below is a simple extension module:

```
from laurelin.ldap import LDAP, LDAPObject

def get_group_members(self, dn):
    """get_group_members(dn)

    Get all members of a group at a particular dn.

    :param str dn: The group's distinguished name
    :return: A list of member usernames
    :rtype: list[str]
    """
    group = self.get(dn)
    return group.get_attr('memberUid')

def obj_get_group_members(self):
    """obj_get_group_members()

    Get all members of this group object.

    :return: A list of member usernames
    :rtype: list[str]
    """
    return self.get_attr('memberUid')

obj_get_group_members.__name__ = 'get_group_members'

def activate_extension()
    LDAP.EXTEND([get_group_members])
    LDAPObject.EXTEND([obj_get_group_members])
```

The module can then be used like so:

```
from laurelin.ldap import LDAP
LDAP.activate_extension('extension.module.name')

with LDAP() as ldap:
    print(ldap.get_group_members('cn=foo,ou=groups,o=example')) # Example LDAP usage
    print(ldap.get('cn=foo,ou=groups,o=example').get_group_members()) # Example_
↪LDAPObject usage
```

5.3 LDAP Extensions

When defining an actual LDAP extension with an OID and requiring server support, you'll create the laurelin extension as shown above, but you'll be calling the `LDAP.send_extended_request()` method from your extension methods.

`LDAP.send_extended_request(oid, value=None, **kws)`

Send an extended request, returns instance of `ExtendedResponseHandle`

This is mainly meant to be called by other built-in methods and client extensions. Requires handling of raw pyasn1 protocol objects.

Parameters

- **oid** (*str*) – The OID of the extension. Must be declared as supported by the server in the root DSE.
- **value** (*str* or *bytes* or *None*) – The request value (optional)

Returns An iterator yielding tuples of the form (`rfc4511.IntermediateResponse`, `rfc4511.Controls`) or (`rfc4511.ExtendedResponse`, `rfc4511.Controls`).

Return type `ExtendedResponseHandle`

Raises

- **`LDAPSupportError`** – if the OID is not listed in the `supportedExtension` attribute of the root DSE
- **`TypeError`** – if the `value` parameter is not a valid type

Additional keyword arguments are handled as `Controls` and then passed through into the `ExtendedResponseHandle` constructor.

As you can see, this accepts the OID of the LDAP extension and an optional request value. You can also pass control keywords, and the `require_success` keyword, which will automatically check for success on the final `extendedResponse` message (and raise an `LDAPError` on failure).

If your LDAP extension expects `intermediateResponse` messages, you can iterate the return from `LDAP.send_extended_request()`. You can also call `ExtendedResponseHandle.recv_response()` to get only one message at a time (preferred to iteration if you only expect the one `extendedResponse` message).

The built-in `LDAP.who_am_i()` method is an excellent example of a simple LDAP extension:

```
from laurelin.ldap import LDAP
from laurelin.ldap.protoutils import get_string_component

def who_am_i(self):
    handle = self.send_extended_request(LDAP.OID_WHOAMI, require_success=True,
    ↪ **ctrl_kws)
    xr, res_ctrls = handle.recv_response()
    return get_string_component(xr, 'responseValue')
```

If this were a laurelin extension, you could go on to bind it to `LDAP` as follows:

```
def activate_extension()
    LDAP.EXTEND([who_am_i])
```

5.4 Controls

Extensions may wish to define controls for use on existing methods. See [Defining Controls](#) for more information.

5.5 Schema

Extensions may be associated with a set of new schema elements, including object classes, attribute types, matching rules, and syntax rules. Once defined, these will get used automatically by other parts of laurelin, including the *SchemaValidator*, and for comparing items in attribute value lists within an *LDAPObject*.

5.5.1 Object Classes and Attribute Types

Creating object classes and attribute types is very simple. Just take the standard LDAP specification and pass it to the appropriate class constructor. Examples from the netgroups extension:

```
from laurelin.ldap.objectclass import ObjectClass
from laurelin.ldap.attributetype import AttributeType

ObjectClass('''
( 1.3.6.1.1.1.2.8 NAME 'nisNetgroup' SUP top STRUCTURAL
  MUST cn
  MAY ( nisNetgroupTriple $ memberNisNetgroup $ description ) )
''')

AttributeType('''
( 1.3.6.1.1.1.1.14 NAME 'nisNetgroupTriple'
  DESC 'Netgroup triple'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.1.1.0.0 )
''')
```

5.5.2 Matching Rules

Defining matching rules takes a little more effort. Matching rules must subclass *EqualityMatchingRule*. Required class attributes include:

- **OID** - the numeric OID of this rule. Note that this does not need to be IANA-registered to work in laurelin, but it still must be globally unique.
- **NAME** - the name of the rule. Must also be globally unique. This is usually how matching rules are referenced in attribute type specs (see `caseExactMatch` in above example).
- **SYNTAX** - the numeric OID of the syntax rule that assertion values must match.

Matching rule classes may also optionally define the following attribute:

- **prep_methods** - a sequence of callables that will be used to prepare both the attribute value and assertion value for comparison. These will typically be defined in `laurelin.ldap.rfc4518`. The initial attribute/assertion value will be passed into the first item in the sequence, and the return from each is passed into the next item.

If you prefer, you can also override the *MatchingRule.prepare()* method on your matching rule class.

You may also wish to override *EqualityMatchingRule.do_match()*. This is passed the two prepared values and must return a boolean. Overriding *MatchingRule.match()* is *not recommended*.

Below is an example matching rule from *laurelin.ldap.schema*:

```

from laurelin.ldap.rules import EqualityMatchingRule
from laurelin.ldap import rfc4518

class numericStringMatch(EqualityMatchingRule):
    OID = '2.5.13.8'
    NAME = 'numericStringMatch'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.36'
    prep_methods = (
        rfc4518.Transcode,
        rfc4518.Map.characters,
        rfc4518.Normalize,
        rfc4518.Prohibit,
        rfc4518.Insignificant.numeric_string,
    )

```

5.5.3 Syntax Rules

Syntax rules must subclass *SyntaxRule*, although in almost all cases you can use *RegexSyntaxRule*. If you do not use a regular expression, you must override *SyntaxRule.validate()*, which receives a single string argument, and must raise *InvalidSyntaxError* when it is incorrect.

In all cases, you must define the following attributes on your syntax rule class:

- **OID** - the numeric OID of the rule. As with matching rules, there is no requirement that this is IANA-registered, but it must be globally unique.
- **DESC** - a brief description of the rule. This is mainly used in exception messages.

Regex syntax rules must also define:

- **regex** - the regular expression.

Below are examples from *laurelin.ldap.schema*:

```

from laurelin.ldap.rules import SyntaxRule, RegexSyntaxRule
from laurelin.ldap.exceptions import InvalidSyntaxError
import six

class DirectoryString(SyntaxRule):
    OID = '1.3.6.1.4.1.1466.115.121.1.15'
    DESC = 'Directory String'

    def validate(self, s):
        if not isinstance(s, six.string_types) or (len(s) == 0):
            raise InvalidSyntaxError('Not a valid {0}'.format(self.DESC))

class Integer(RegexSyntaxRule):
    OID = '1.3.6.1.4.1.1466.115.121.1.27'
    DESC = 'INTEGER'
    regex = r'^-?[1-9][0-9]*$'

```

5.5.4 SchemaValidator

Laurelin ships with *SchemaValidator* which, when applied to a connection, automatically checks write operations for schema validity *before* sending the request to the server. This includes any schema you define in your extensions. Users can enable this like so:

```
from laurelin.ldap import LDAP
from laurelin.ldap.schema import SchemaValidator

with LDAP('ldaps://dir.example.org', validators=[SchemaValidator()]) as ldap:
    # do stuff
```

You can also define your own validators, see below.

5.6 Validators

Validators must subclass *Validator*. The public interface includes *Validator.validate_object()* and *Validator.validate_modify()*. You will usually just want to override these, however they do include a default implementation which checks all attributes using the abstract *Validator._validate_attribute()*. Check method docs for more information about how to define these.

When defining validators in your extension, you can avoid needing to import the module again by using the return value from *LDAP.activate_extension()*, like so:

```
from laurelin.ldap import LDAP
my_ext = LDAP.activate_extension('an.extension.module')

with LDAP('ldaps://dir.example.org', validators=[my_ext.MyValidator()]) as ldap:
    # do stuff
```

5.7 Packaging

laurelin.extensions is a *namespace package* meaning you can add your own modules and packages to it. You can use this on your private infrastructure, publish it in its own package that way, or submit it as a pull request to be shipped as a built-in extension. You're also welcome to package in your own namespace, as long as it is reachable for import.

- *Using Controls*
- *Defining Controls*

Many LDAP users may be unfamiliar with controls. RFC4511 defines *controls* as “providing a mechanism whereby the semantics and arguments of existing LDAP operations may be extended.” In other words, they can:

1. Instruct the server to process a method differently
2. Add new arguments to methods to control the altered processing
3. Add additional data to the response to a method call

It is important to note that both the server and client must mutually support all controls used. Laurelin will automatically check for server support when using controls.

6.1 Using Controls

Once controls have been *defined*, they are very easy to use. Each control has a keyword and optionally a `response_attr`.

The keyword can be passed as a keyword argument to specific methods. The value type and format is up to the control implementation. Whatever value the control expects can be wrapped in *critical* or *optional* to declare the criticality of the control.

If defined, the `response_attr` will be set as an attribute on the object returned from the method call.

For search response controls, the control value will be set on the individual *LDAPObject* if it appeared on the associated search result entry. If it appeared on the search results done message, the control value will be set on the iterator object.

In the highly unusual case that a response control is set on a search result reference message, the control values will be inaccessible if `fetch_result_refs` is set to `True`. A warning will be issued in this case.

If `fetch_result_refs` is set to `False`, the response control values will be set on the `SearchReferenceHandle` that is yielded from the results iterator.

```
class laurelin.ldap.critical(value)
```

Bases: `object`

used to mark controls with criticality

```
class laurelin.ldap.optional(value)
```

Bases: `object`

used to mark controls as not having criticality

An `LDAPSupportError` will be raised if the control is marked critical and the server does not support it.

6.2 Defining Controls

Controls must subclass `Control`. As soon as they are defined as a subclass of `Control`, they are ready to use. Controls must define at least:

- `Control.method`, a tuple of method names that this control supports. Current method names are `bind`, `search`, `compare`, `add`, `delete`, `mod_dn`, `modify`, and `ext` (extended request). Note that these method names do not necessarily correspond directly to `LDAP` method names. Even when they do, other methods may call the base method and pass through control keywords. For example, `LDAPObject.find()` ends up passing any control keywords through into `LDAP.search()` (which matches the `search` method). The `bind` method is used by both `LDAP.simple_bind()` and `LDAP.sasl_bind()`.
- `Control.keyword`, the keyword argument to be used for the request control.
- `Control.REQUEST_OID` the OID of the request control. If the control has criticality, the OID must be listed in the `supportedControl` attribute of the root DSE of the server at runtime.

If there is an associated response control, also define the following:

- `Control.response_attr`, the name of the attribute which will be set on objects returned from the method.
- `Control.RESPONSE_OID` the OID of the response control. This may be equal to `Control.REQUEST_OID` depending on the spec. This must match the `controlType` of the response control to be properly set.

Most controls will not need to override methods if only strings are used for request and response values. However, if it is desirable to use a more complex data structure as a control value, you can override the `Control.prepare()` method to accept this structure as its first argument. You will need to process this into a single string for transmission to the server, and pass it into, and return, the base `Control.prepare()`. The second argument is a boolean describing criticality, and must also be passed into the base method.

To return a more complex value for the response, you can override the `Control.handle()` method. This will be passed the response control value string, and the return will be assigned to the `response_attr` attribute on the returned object.

```
class laurelin.ldap.controls.Control
```

Bases: `object`

Request controls are exposed by allowing an additional keyword argument on a set of methods. The `prepare()` method takes the value passed in as a keyword argument and returns an `rfc4511.Control`.

Response controls are returned by setting an additional attribute on whichever object is returned by the called method. The raw response control value is passed to the `handle()` method, and any appropriate value may be returned.

Leave the `RESPONSE_OID` and `response_attr` attributes as a `False` value if there is no response control specified.

REQUEST_OID = ''

Request OID of the control

RESPONSE_OID = ''

Response OID of the control (may be equal to `REQUEST_OID`; may be left empty)

handle (*ctrl_value*)

Accepts raw response *ctrl_value* and may return any useful value.

There is no need to call this base function when overriding.

Parameters **ctrl_value** (*str*) – The string response control value received from the server.

Returns The string *ctrl_value* unchanged by default. May be overridden to return any relevant value/type/structure.

keyword = ''

keyword argument name

method = ()

name(s) of the method which this control is used with

prepare (*ctrl_value*, *criticality*)

Accepts string controlValue and returns an `rfc4511.Control` instance

When overriding this function, you must always call and return this base function.

Parameters

- **ctrl_value** (*str or bytes*) – The string request control value to send to the server
- **criticality** (*bool*) – True if the control has criticality. This is indicated by wrapping the keyword argument in `critical` or `optional`, and by the *default_criticality* keyword passed to the `LDAP` constructor, and global default `LDAP.DEFAULT_CRITICALITY`.

Returns The protocol-level control object ready for transmission to the server

Return type `rfc4511.Control`

response_attr = ''

Name of the attribute where return of `handle()` will be stored

7.1 1.5.3

Release 2018.08.30

- Add python 3.7 support

7.2 1.5.2

Released 2018.06.15

1.5.1 was built off of the wrong branch and will be removed.

- Minor fix: Added FilterSyntax to all
- Doc update: added dependent info section to readme

7.3 1.5.0

Released 2018.06.09

- Added new simple filter syntax
- Switched default filter syntax to UNIFIED which should be backwards compatible with standard RFC 4515 filters

Special thanks to @jpypi for authoring the new grammar

7.4 1.4.1

Released 2018.05.31

- Fix: Checked for failed import of AF_UNIX to improve Windows support
- Fix: Required latest pure-sasl

7.5 1.4.0

Released 2018.05.29

- Validation updates:
 - Added `LDAP.disable_validation()` which creates a context with any or all validators skipped
 - Added an `ldap_conn` attribute to validator instances to allow validators to query the server
 - Allowed passing a class as well as an instance with the `validators` constructor keyword
- Greatly improved handling of unsolicited messages (message ID 0)
- Fix: enforce maximum length for attribute types
- Fix: SASL auth issues with pure-sasl 0.5.1+

7.6 1.3.1

Released 2018.04.01

- Fixed logic bug in `SchemaValidator` when an object has two or more object classes that require one or more of the same attributes
- Fixed: allowed string `some.module.Class` specification for validators in config files

7.7 1.3.0

Released 2018.03.22

- Added config file support, see `laurelin.ldap.config`
- Fixed: ensured extensions can be safely activated multiple times
- Fixed: `Mod` constants `repr` updated for consistency

7.8 1.2.0

Released 2018.03.16

- Add `DELETE_ALL` to use as an attribute value list with `modify`, `replace_attrs`, and `delete_attrs`
- Added new constructor keywords to alter the behavior of empty value lists for `modify`, `replace_attrs`, and `delete_attrs`:
 - `ignore_empty_list` to silently ignore empty value lists and not send them to the server. This will be enabled by default in a future release.
 - `error_empty_list` to raise an exception when an empty value list is passed.
 - `warn_empty_list` to emit a warning when an empty value list is passed.

7.9 1.1.0

Released 2018.03.12

Initial stable API.

8.1 laurelin.Ldap package

8.1.1 Submodules

8.1.2 laurelin.Ldap.attributetype module

class `laurelin.ldap.attributetype.AttributeType` (*spec*)

Bases: `object`

Parses an LDAP attribute type specification and implements supertype inheritance.

Each instantiation registers the names and OIDs specified so that the `spec` can be accessed using `get_attribute_type()`.

See the `laurelin.ldap.schema` module source for example usages.

Parameters `spec` (*str*) – The LDAP specification for an Attribute Type.

Raises `LDAPSchemaError`: * if the specification is invalid * if the OID has already been defined * if one of the names has already been defined

Variables

- `oid` (*str*) – The OID of the attribute type
- `names` (*tuple* (*str*)) – A tuple containing all possible names for the attribute type
- `supertype` (*str*) – The specified supertype. If the `spec` does not define optional properties, they will pass through into the supertype.
- `equality_oid` (*str*) – The OID of the equality matching rule
- `syntax_oid` (*str*) – The OID of the syntax matching rule
- `syntax_length` (*int*) – The suggested maximum length of a value

- **obsolete** (*bool*) – The type has been flagged as obsolete. Will cause a warning from the `SchemaValidator` if an obsolete attribute type is used.
- **single_value** (*bool*) – The attribute may only have one value.
- **collective** (*bool*) – The attribute has been marked collective.
- **no_user_mod** (*bool*) – The attribute may not be modified by users (e.g., for operational attributes). Will cause a validation failure from the `SchemaValidator` if a write operation is attempted on attribute types with this property set to `True`.
- **usage** (*str*) – A string describing the attribute’s usage. May be one of *userApplications*, *directoryOperation*, *distributedOperation*, or *dSAOperation*.

equality

Gets the `EqualityMatchingRule` for this attribute type.

index (*value_list*, *assertion_value*)

Finds the index of a value in a list of attribute values. Raises a `ValueError` if the value is not found in the list. Assumes values in *value_list* are already validated.

Parameters

- **value_list** (*list[str]*) – The list of attribute values. Assumes values are already validated.
- **assertion_value** (*str*) – The value to look for in *value_list*.

Returns The index of *assertion_value* in *value_list*.

Return type `int`

Raises

- **ValueError** – if *assertion_value* is not found or if *value_list* is empty.
- **InvalidSyntaxError** – if *assertion_value* does not meet the syntax requirements of this attribute type

syntax

Gets the `SyntaxRule` for this attribute type.

validate (*value*)

Validate a value according to the attribute type’s syntax rule.

Parameters **value** (*str*) – The potential attribute value

Returns A truthy value.

Raises **InvalidSyntaxError** – if the value is invalid.

class `laurelin.ldap.attributetype.DefaultAttributeType` (*name=None*)

Bases: `laurelin.ldap.attributetype.AttributeType`

The default attribute type returned by `get_attribute_type()` when the requested attribute type is undefined.

Essentially behaves as an unrestricted case-sensitive attribute type.

Users should probably never instantiate this.

equality

Gets the `EqualityMatchingRule` for this attribute type.

index (*value_list*, *assertion_value*)

Finds the index of a value in a list of attribute values. Raises a `ValueError` if the value is not found in the list. Assumes values in *value_list* are already validated.

Parameters

- **value_list** (*list* [*str*]) – The list of attribute values. Assumes values are already validated.
- **assertion_value** (*str*) – The value to look for in *value_list*.

Returns The index of *assertion_value* in *value_list*.

Return type `int`

Raises

- **ValueError** – if *assertion_value* is not found or if *value_list* is empty.
- **InvalidSyntaxError** – if *assertion_value* does not meet the syntax requirements of this attribute type

syntax

Gets the `SyntaxRule` for this attribute type.

class `laurelin.ldap.attributetype.DefaultMatchingRule`

Bases: `object`

The default matching rule to use for undefined attribute types.

Users should probably never instantiate this.

do_match (*a*, *b*)

Require strict equality

match (*a*, *b*)

Do the match

prepare (*a*)

Do nothing to prepare

validate (*value*)

Allow all values

class `laurelin.ldap.attributetype.DefaultSyntaxRule`

Bases: `object`

The default syntax rule to use for undefined attribute types.

Users should probably never instantiate this.

validate (*s*)

Allow all values

`laurelin.ldap.attributetype.get_attribute_type` (*ident*)

Get an instance of `AttributeType` associated with either a name or OID.

Parameters **ident** (*str*) – Either the numeric OID of the desired attribute type spec or any one of its specified names

Returns The `AttributeType` containing a parsed specification

Return type `AttributeType`

8.1.3 laurelin.ldap.attrsdict module

```
class laurelin.ldap.attrsdict.AttrsDict (attrs_dict=None)
    Bases: laurelin.ldap.utils.CaseIgnoreDict

    Stores attributes and provides utility methods without any server or object affinity

    Dict keys are case-insensitive attribute names, and dict values are a list of attribute values

    deepcopy ()
        Return a native dict copy of self.

    get_attr (attr)
        Get an attribute's values, or an empty list if the attribute is not defined

        Parameters attr – The name of the attribute

        Returns A list of values

        Return type list

    iterattrs ()
        Iterate all attributes of this object. Yields (attr, value) tuples.

    setdefault (k, d) → D.get(k,d), also set D[k]=d if k not in D

    update ([E], **F) → None. Update D from dict/iterable E and F.
        If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a
        .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

    static validate (attrs_dict)
        Validate that attrs_dict is either already an AttrsDict or that it conforms to the required
        dict(str, list(str or bytes)) typing.

        Parameters attrs_dict (dict) – The dictionary to validate for use as an attributes dictionary

        Return type None

        Raises TypeError – when the dict is invalid

    static validate_attr (attr)
        Validate that attr is a valid attribute name.

        Parameters attr (str) – The string to validate for use as an attribute name

        Return type None

        Raises TypeError – when the string is invalid

    static validate_values (attr_val_list)
        Validate that attr_val_list conforms to the required list(str or bytes) typing. Also allows
        the DELETE_ALL constant.

        Parameters attr_val_list (list) – The list to validate for use as an attribute value list.

        Return type None

        Raises TypeError – when the list is invalid
```

8.1.4 laurelin.ldap.attrvaluelist module

```
class laurelin.ldap.attrvaluelist.AttrValueList (attr, values)
    Bases: list
```

List that follows schema matching rules for the `in` operator and other related methods.

Parameters

- **attr** (*str*) – The attribute name or type identifier
- **values** (*list[str]*) – Initial values for the list

count (*value*)

Count the number of occurrences of *value*. Since attribute value lists are defined to only have at most one unique copy of any value, this will always return 0 or 1.

Parameters **value** (*str*) – The value to count

Returns The number of occurrences of *value*, 1 or 0.

Return type `int`

Raises **ValueError** – if the value is not found or if the list has no values

index (*value*, **args*, ***kwds*)

Find the index of *value* or raise a `ValueError` if not found. The stock start/end arguments are ignored since a list of attribute values is defined to have exactly zero or one unique matching values.

Parameters **value** (*str*) – The value to find

Returns The index of the value

Return type `int`

Raises **ValueError** – if the value is not found or if the list has no values

remove (*value*)

Remove *value* from the list if present.

Parameters **value** (*str*) – The value to remove

Return type `None`

Raises **ValueError** – if the value is not found or if the list has no values

8.1.5 laurelin.ldap.base module

Contains base classes for `laurelin.ldap`

class `laurelin.ldap.base.CompareResponse` (*compare_result*)

Bases: `laurelin.ldap.base.LDAPResponse`

Stores boolean compare result and any response control values. The `bool()` of this object gives the compare result.

class `laurelin.ldap.base.ExtendedResponseHandle` (*mid*, *ldap_conn*, *require_success=False*)

Bases: `laurelin.ldap.base.ResponseHandle`

Obtains `rfc4511.ExtendedResponse` or `rfc4511.IntermediateResponse` instances from the server for a particular message ID

recv_response ()

```
class laurelin.ldap.base.LDAP(server=None, base_dn=None, reuse_connection=None, connect_timeout=None, search_timeout=None, deref_aliases=None, strict_modify=None, ssl_verify=None, ssl_ca_file=None, ssl_ca_path=None, ssl_ca_data=None, fetch_result_refs=None, default_sasl_mech=None, sasl_fatal_downgrade_check=None, default_criticality=None, follow_referrals=None, validators=None, warn_empty_list=None, error_empty_list=None, ignore_empty_list=None, filter_syntax=None)
```

Bases: `laurelin.ldap.extensible.Extensible`

Provides the connection to the LDAP DB. All constructor parameters have a matching global default as a class property on `LDAP`

Parameters

- **server** (*str* or `LDAPSock`) – URI string to connect to or an `LDAPSock` to reuse
- **base_dn** (*str*) – The DN of the base object
- **reuse_connection** (*bool*) – Allows the socket connection to be reused and reuse an existing socket if possible.
- **connect_timeout** (*int*) – Number of seconds to wait for connection to be accepted.
- **search_timeout** (*int*) – Number of seconds to wait for a search to complete. Partial results will be returned when the timeout is reached. Can be overridden on a per-search basis by setting the `search_timeout` keyword on `LDAP.search()`.
- **deref_aliases** (`DerefAliases`) – One of the `DerefAliases` constants. Instructs the server how to handle alias objects in search results. Can be overridden on a per-search basis by setting the `deref_aliases` keyword on `LDAP.search()`.
- **strict_modify** (*bool*) – Use the strict modify strategy. If set to True, guarantees that another search will not take place before a modify operation. May potentially produce more server errors.
- **ssl_verify** (*bool*) – Validate the certificate and hostname on an SSL/TLS connection
- **ssl_ca_file** (*str*) – Path to PEM-formatted concatenated CA certificates file
- **ssl_ca_path** (*str*) – Path to directory with CA certs under hashed file names. See https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_load_verify_locations.html for more information about the format of this directory.
- **ssl_ca_data** (*str* or *bytes*) – An ASCII string of one or more PEM-encoded certs or a bytes object containing DER-encoded certificates.
- **fetch_result_refs** (*bool*) – Fetch `searchResultRef` responses in search results. Can be overridden on a per-search basis by setting the `fetch_result_refs` keyword on `LDAP.search()`.
- **default_sasl_mech** (*str*) – Name of the default SASL mechanism. Bind will fail if the server does not support the mechanism. (Examples: DIGEST-MD5, GSSAPI)
- **sasl_fatal_downgrade_check** (*bool*) – Set to False to make potential downgrade attack check non-fatal.
- **default_criticality** (*bool*) – Set to True to make controls critical by default, set to False to make non-critical
- **follow_referrals** (*bool*) – Automatically follow referral results

- **validators** (*list[Validator]*) – A list of `Validator` instances to apply to this connection.
- **warn_empty_list** (*bool*) – Default `False`. Set to `True` to emit a warning when an empty value list is passed to `LDAP.modify()`, `LDAP.replace_attrs()`, or `LDAP.delete_attrs()` or their `LDAPObject` counterparts.
- **error_empty_list** (*bool*) – Default `False`. Set to `True` to raise an exception when an empty value list is passed to `LDAP.modify()`, `LDAP.replace_attrs()`, or `LDAP.delete_attrs()` or their `LDAPObject` counterparts.
- **ignore_empty_list** (*bool*) – Default `False`. Set to `True` to ignore empty value lists passed to `LDAP.modify()`, `LDAP.replace_attrs()`, or `LDAP.delete_attrs()` or their `LDAPObject` counterparts. This will be default `True` in a future release.
- **filter_syntax** (*FilterSyntax*) – The default search filter syntax selection. Must be one of the `FilterSyntax` constants. Can be overridden on a per-search basis by setting the `filter_syntax` keyword on `LDAP.search()`. Defaults to `FilterSyntax.STANDARD` for RFC4515-compliant filter string syntax.

The class can be used as a context manager, which will automatically unbind and close the connection when the context manager exits.

Example:

```
with LDAP() as ldap:
    raise Exception()
# ldap is closed and unbound

with LDAP() as ldap:
    print('hello')
# ldap is closed and unbound
```

```
DEFAULT_BASE_DN = None
DEFAULT_CONNECT_TIMEOUT = 5
DEFAULT_CRITICALITY = False
DEFAULT_DEREF_ALIASES = DerefAliases.ALWAYS
DEFAULT_ERROR_EMPTY_LIST = False
DEFAULT_FETCH_RESULT_REFS = True
DEFAULT_FILTER = '(objectClass=*)'
DEFAULT_FILTER_SYNTAX = FilterSyntax.UNIFIED
DEFAULT_FOLLOW_REFERRALS = True
DEFAULT_IGNORE_EMPTY_LIST = False
DEFAULT_REUSE_CONNECTION = True
DEFAULT_SASL_FATAL_DOWNGRADE_CHECK = True
DEFAULT_SASL_MECH = None
DEFAULT_SEARCH_TIMEOUT = 0
DEFAULT_SERVER = 'ldap://localhost'
DEFAULT_SSL_CA_DATA = None
```

```
DEFAULT_SSL_CA_FILE = None
```

```
DEFAULT_SSL_CA_PATH = None
```

```
DEFAULT_SSL_VERIFY = True
```

```
DEFAULT_STRICT_MODIFY = False
```

```
DEFAULT_VALIDATORS = None
```

```
DEFAULT_WARN_EMPTY_LIST = False
```

```
DELETE_ALL = <delete all values>
```

Use with modify replace/delete in place of an attribute list to delete all values for the attribute

```
LOG_FORMAT = "[% (asctime)s] %(name)s %(levelname)s : %(message)s "
```

```
NO_ATTRS = '1.1'
```

```
OID_OBJ_CLASS_ATTR = '1.3.6.1.4.1.4203.1.5.2'
```

```
OID_STARTTLS = '1.3.6.1.4.1.1466.20037'
```

```
OID_WHOAMI = '1.3.6.1.4.1.4203.1.11.3'
```

```
static activate_extension(module_name)
```

Import the module name and call the `activate_extension` function on the module.

Parameters `module_name` (*str*) – The name of the module to import and activate

Returns The imported module

Return type module

```
add(dn, attrs_dict, **kws)
```

Add new object and return corresponding LDAPObject on success.

Parameters

- `dn` (*str*) – The new object's DN
- `attrs_dict` (*dict*(*str*, *list*[*str* or *bytes*]) or *AttrsDict*) – The new attributes for the object

Returns The new object

Return type *LDAPObject*

Raises

- *ConnectionUnbound* – if the connection has been unbound
- *TypeError* – if arguments are of invalid type
- *LDAPValidationError* – if the object fails any configured validator
- *LDAPError* – if we get a non-success result

Additional keyword arguments are handled as *Controls* and then passed through into `LDAP.obj()`.

```
add_attrs(dn, attrs_dict, current=None, **ctrl_kws)
```

Add new attribute values to existing object.

Parameters

- `dn` (*str*) – The DN of the object to modify
- `attrs_dict` (*dict*(*str*, *list*[*str* or *bytes*]) or *AttrsDict*) – The new attributes to add to the object

- **current** (`LDAPObject` or `None`) – The current known state of the object. Used for ensuring we don't send duplicate attributes to the server and for validation.

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

add_if_not_exists (*dn*, *attrs_dict*)

Add object if it doesn't exist

- Gets and returns the object at DN if it exists, otherwise create the object using the attrs dictionary
- Always returns an `LDAPObject` corresponding to the final state of the DB

Parameters

- **dn** (*str*) – The object DN
- **attrs_dict** (*dict(str, list[str or bytes])* or *AttrsDict*) – The attributes to use if adding the object

Returns The new or existing object

Return type *LDAPObject*

add_or_mod_add_if_exists (*dn*, *attrs_dict*)

Add object if it doesn't exist, otherwise add_attrs

- If the object at DN exists, perform an add modification using the attrs dictionary. Otherwise, create the object using the attrs dictionary.
- This ensures that, for the attributes mentioned in attrs, AT LEAST those values will exist on the given DN, regardless of prior state of the DB.
- Always returns an `LDAPObject` corresponding to the final state of the DB

Parameters

- **dn** (*str*) – The object DN
- **attrs_dict** (*dict(str, list[str or bytes])* or *AttrsDict*) – The objects minimum attributes

Returns The new or modified object

Return type *LDAPObject*

add_or_mod_replace_if_exists (*dn*, *attrs_dict*)

Add object if it doesn't exist, otherwise replace_attrs

- If the object at DN exists, perform a replace modification using the attrs dictionary. Otherwise, create the object using the attrs dictionary
- This ensures that, for the attributes mentioned in attrs, ONLY those values will exist on the given DN regardless of prior state of the DB.
- Always returns an `LDAPObject` corresponding to the final state of the DB

Parameters

- **dn** (*str*) – The object DN

- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The objects new required attributes

Returns The new or modified object

Return type *LDAPObject*

close (*force=False*)

Send an unbind request and close the socket.

Parameters **force** (*bool*) – Unbind and close the socket even if other objects still hold a reference to it.

Raises *ConnectionUnbound* – if the connection has already been unbound

compare (*dn, attr, value, **ctrl_kwds*)

Ask the server if a particular DN has a matching attribute value. The comparison will take place following the schema-defined matching rules and syntax rules.

Parameters

- **dn** (*str*) – The DN of the object
- **attr** (*str*) – The attribute name
- **value** (*str*) – The assertion value

Returns A response object, *bool()* evaluating to the result of the comparison

Return type *CompareResponse*

Raises

- *ConnectionUnbound* – if the connection has been unbound
- *LDAPError* – if we got a result other than compareTrue or compareFalse

Additional keyword arguments are handled as *Controls*.

static default_warnings ()

Always take the default action for warnings

delete (*dn, **ctrl_kwds*)

Delete an object.

Parameters **dn** (*str*) – The DN of the object to delete

Returns A response object

Return type *LDAPResponse*

Raises *ConnectionUnbound* – if the connection has been unbound

Additional keyword arguments are handled as *Controls*.

delete_attrs (*dn, attrs_dict, current=None, **ctrl_kwds*)

Delete specific attribute values from *attrs_dict*.

Specifying a 0-length entry will delete all values.

Parameters

- **dn** (*str*) – The DN of the object to modify
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The attributes to remove from the object. Specify an empty list for a value to delete all values.

- **current** (`LDAPObject` or `None`) – The current known state of the object. Used to ensure we don't request that the server delete attribute values that don't exist and for validation.

Returns A response object

Return type `LDAPResponse`

Additional keyword arguments are handled as *Controls*.

disable_validation (*disabled_validators=None*)

Returns a context manager which temporarily disables validation. If any server errors are generated, they will still be propagated.

Example:

```
from laurelin.ldap import LDAP
from laurelin.ldap.exceptions import LDAPValidationError
from laurelin.ldap.schema import SchemaValidator

with LDAP(validators=[SchemaValidator()]) as ldap:
    # make validated queries
    ldap.base.add_child('cn=foo', {<valid object>})

    try:
        ldap.base.add_child('cn=bar', {<invalid object>})
    except LDAPValidationError:
        pass

    with ldap.disable_validation(['SchemaValidator']):
        # make queries without validation
        ldap.base.add_child('cn=bar', {<invalid object>})
        # NOTE: if the object is actually invalid, a server error may still_
        ↳ occur

    # carry on with validation restored...
```

Parameters disabled_validators – Optional, a list of string class names or Validator classes to disable. By default all validators will be disabled.

Returns A context manager which temporarily disables validation

Return type `DisabledValidationContext`

static disable_warnings ()

Prevent all LDAP warnings from being shown - default action for others

static enable_logging (*level=10*)

Enable logging output to stderr

exists (*dn*)

Simply check if a DN exists.

Parameters dn (*str*) – The DN to check

Returns True if the object exists, False if not

Return type `bool`

get (*dn, attrs=None, **kws*)

Get a specific object by DN.

Performs a search with `Scope.BASE` and ensures we get exactly one result.

Parameters

- **dn** (*str*) – The DN of the object to query
- **attrs** (*list[str]* or *None*) – Optional. A list of attribute names to get, defaults to all user attributes

Returns The LDAP object

Return type *LDAPObject*

Raises

- *ConnectionUnbound* – if the connection has been unbound
- *NoSearchResults* – if no results are returned
- *MultipleSearchResults* – if more than one result is returned

Additional keyword arguments are passed through into *LDAP.search()*.

get_sasl_mechs()

Query root DSE for supported SASL mechanisms.

Returns The list of server-supported mechanism names.

Return type *list[str]*

static log_warnings()

Log all LDAP warnings rather than showing them - default action for others

mod_dn (*dn*, *new_rdn*, *clean_attr=True*, *new_parent=None*, ***ctrl_kwds*)

Change the DN and possibly the location of an object in the tree. Exposes all options of the protocol-level *rfc4511.ModifyDNRequest*

Parameters

- **dn** (*str*) – The current DN of the object
- **new_rdn** (*str*) – The new RDN of the object, e.g. *cn=foo*
- **clean_attr** (*bool*) – Remove the old RDN attribute from the object when changing
- **new_parent** (*str* or *None*) – The DN of the new parent object, or *None* to leave the location unchanged

Returns A response object

Return type *LDAPResponse*

Raises *ConnectionUnbound* – if the connection has been unbound

Additional keyword arguments are handled as *Controls*.

modify (*dn*, *modlist*, *current=None*, ***ctrl_kwds*)

Perform a series of modify operations on an object atomically

Parameters

- **dn** (*str*) – The DN of the object to modify
- **modlist** (*list[Mod]*) – A list of *Mod* instances, e.g. [*Mod*(*Mod.ADD*, 'someAttr', ['value1', 'value2'])]
- **current** (*LDAPObject* or *None*) – The current known state of the object for use in validation

Returns A response object

Return type *LDAPResponse*

Raises

- *ConnectionUnbound* – if the connection has been unbound
- *LDAPValidationError* – if the operation fails and configured validator

Additional keyword arguments are handled as *Controls*.

move (*dn*, *new_dn*, *clean_attr=True*, ***ctrl_kwds*)

Specify a new absolute DN for an object.

Parameters

- **dn** (*str*) – The current DN of the object
- **new_dn** (*str*) – The new absolute DN of the object, e.g. cn=foo,dc=example,dc=org
- **clean_attr** (*bool*) – Remove the old RDN attribute from the object when changing

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

obj (*dn*, *attrs_dict=None*, *tag=None*, ***kwds*)

Factory for LDAPObjects bound to this connection.

Note that this does not query the server. Use *LDAP.get()* to query the server for a particular DN.

Parameters

- **dn** (*str*) – The DN of the object.
- **attrs_dict** (*dict(str, list[str or bytes])* or *AttrsDict* or *None*) – Optional. The object's attributes and values.
- **tag** (*str* or *None*) – Optional. The tag for this object. Tagged objects can be retrieved with *LDAP.tag()*.

Returns The new object bound to this connection.

Return type *LDAPObject*

Raises *TagError* – if the tag parameter is already defined

Additional keywords are passed through into the *LDAPObject* constructor.

process_ldif (*ldif_str*)

Process a basic LDIF

TODO: full RFC 2849 implementation. Missing:

- attribute options

Parameters **ldif_str** (*str*) – An RFC 2849 complying LDIF string

Returns A list with elements corresponding to the return of each described operation

Return type *list[LDAPResponse or LDAPObject]*

Raises

- *ValueError* – if the LDIF is malformed

- ***LDAPError*** – if an unimplemented feature is used
- ***LDAPSupportError*** – if a version other than 1 is specified or a critical control is undefined

recheck_sasl_mechs ()

Query the root DSE again after performing a SASL bind to check for a downgrade attack.

Raises ***LDAPError*** – If the downgrade attack check fails and `sasl_fatal_downgrade_check` has not been set to `False`.

refresh_root_dse ()

Update the local copy of the root DSE, containing metadata about the directory server. The root DSE is an `LDAPObject` stored on the `root_dse` attribute.

rename (*dn*, *new_rdn*, *clean_attr=True*, ***ctrl_kwds*)

Specify a new RDN for an object without changing its location in the tree.

Parameters

- ***dn*** (*str*) – The current DN of the object
- ***new_rdn*** (*str*) – The new RDN of the object, e.g. `cn=foo`
- ***clean_attr*** (*bool*) – Remove the old RDN attribute from the object when changing

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

replace_attrs (*dn*, *attrs_dict*, *current=None*, ***ctrl_kwds*)

Replace all values on given attributes with the passed values

- Attributes not mentioned in `attrsDict` are not touched
- Attributes will be created if they do not exist
- Specifying a 0-length entry will delete all values for that attribute

Parameters

- ***dn*** (*str*) – The DN of the object to modify
- ***attrs_dict*** (*dict*(*str*, *list*(*str* or *bytes*)) or *AttrsDict*) – The new attributes to set on the object
- ***current*** (*LDAPObject* or *None*) – The current known state of the object for use in validation

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

sasl_bind (*mech=None*, ***props*)

Perform a SASL bind operation.

Keywords are first taken as *Controls*. Required keyword args are dependent on the mechanism chosen.

Parameters ***mech*** (*str*) – The SASL mechanism name to use or `None` to negotiate best mutually supported mechanism.

Returns A response object

Return type *LDAPResponse*

Raises

- *ConnectionUnbound* – if the connection has been unbound/closed
- *ConnectionAlreadyBound* – if the connection has already been bound
- *LDAPSupportError* – if the given mech is not supported by the server
- *LDAPError* – if an error occurs during the bind process

search (*base_dn*, *scope*=*Scope.SUB*, *filter*=*None*, *attrs*=*None*, *search_timeout*=*None*, *limit*=*0*, *deref_aliases*=*None*, *attrs_only*=*False*, *fetch_result_refs*=*None*, *follow_referrals*=*None*, *filter_syntax*=*None*, ***kws*)

Sends search and return an iterator over results.

Parameters

- **base_dn** (*str*) – The DN of the base object of the search
- **scope** (*Scope*) – One of the *Scope* constants, default *Scope.SUB*. Controls the maximum depth of the search.
- **filter** (*str*) – A filter string. Objects must match the filter to be included in results. Default includes all objects and can be overridden globally by defining *LDAP.DEFAULT_FILTER*.
- **attrs** (*list[str]*) – A list of attribute names to include for each object. Default includes all user attributes. Use ['*', '+'] to get all user and all operational attributes.
- **search_timeout** (*int*) – The number of seconds the server should spend performing the search. Partial results will be returned if the server times out. The default can be set per connection by passing the *search_timeout* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SEARCH_TIMEOUT*.
- **limit** (*int*) – The maximum number of objects to return.
- **deref_aliases** (*DerefAliases*) – One of the *DerefAliases* constants. This instructs the server what to do when it encounters an alias object. The default can be set per connection by passing the *deref_aliases* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_DEREF_ALIASES*.
- **attrs_only** (*bool*) – Default *False*. Set to *True* to only obtain attribute names and not any attribute values.
- **fetch_result_refs** (*bool*) – When the server returns a result which is a reference to an object on another server, automatically attempt to fetch the remote object and include it in the iterated results. The default can be set per connection by passing the *fetch_result_refs* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_FETCH_RESULT_REFS*.
- **follow_referrals** (*bool*) – When the server knows that the base object is present on another server, follow the referral and perform the search on the other server. The default can be set per connection by passing the *follow_referrals* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_FOLLOW_REFERRALS*.
- **filter_syntax** (*FilterSyntax*) – Select which filter syntax to use to parse the filter. The default can be set per connection by passing the *default_filter_syntax* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_FILTER_SYNTAX*.

Returns An iterator over the results of the search. May yield *LDAPObject* or possibly *SearchReferenceHandle* if *fetch_result_refs* is *False*.

Additional keywords are handled as *Controls* first and then passed through into `LDAP.obj()`.

This method may also be used as a context manager. If all results have not been read, the operation will automatically be abandoned when the context manager exits. You can also raise `Abandon` to abandon all results immediately and cleanly exit the context manager. You can also call `SearchResultHandle.abandon()` to abandon results.

Example:

```
# Dump the whole tree
with LDAP() as ldap:
    with ldap.base.search() as search:
        for result in search:
            print(result.format_ldif())
```

send_extended_request (*oid*, *value=None*, ***kwds*)

Send an extended request, returns instance of *ExtendedResponseHandle*

This is mainly meant to be called by other built-in methods and client extensions. Requires handling of raw pyasn1 protocol objects.

Parameters

- **oid** (*str*) – The OID of the extension. Must be declared as supported by the server in the root DSE.
- **value** (*str* or *bytes* or *None*) – The request value (optional)

Returns An iterator yielding tuples of the form (`rfc4511.IntermediateResponse`, `rfc4511.Controls`) or (`rfc4511.ExtendedResponse`, `rfc4511.Controls`).

Return type *ExtendedResponseHandle*

Raises

- **LDAPSupportError** – if the OID is not listed in the `supportedExtension` attribute of the root DSE
- **TypeError** – if the *value* parameter is not a valid type

Additional keyword arguments are handled as *Controls* and then passed through into the *ExtendedResponseHandle* constructor.

simple_bind (*username=""*, *password=""*, ***ctrl_kwds*)

Performs a simple bind operation

Leave arguments as their default (empty strings) to attempt an anonymous simple bind

Additional keywords are used as *Controls*.

Parameters

- **username** (*str*) – Bind DN/username or empty string for anonymous
- **password** (*str*) – Password to bind with or empty string for anonymous

Returns A response object

Return type *LDAPResponse*

Raises

- **ConnectionUnbound** – if the connection has been unbound/closed
- **ConnectionAlreadyBound** – if the connection has already been bound

start_tls (*verify=None, ca_file=None, ca_path=None, ca_data=None*)

Perform the StartTLS extended operation. This will instruct the server to begin encrypting this socket connection with TLS/SSL.

Parameters

- **verify** (*bool*) – Set to False to disable verification of the remote certificate. You can set the default per-connection by passing the *ssl_verify* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_VERIFY*.
- **ca_file** (*str*) – Path to PEM-formatted concatenated CA certificates file. You can set the default per-connection by passing the *ssl_ca_file* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_CA_FILE*.
- **ca_path** (*str*) – Path to directory with CA certs under hashed file names. See https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_load_verify_locations.html for more information about the format of this directory. You can set the default per-connection by passing the *ssl_ca_path* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_CA_PATH*.
- **ca_data** (*str or bytes*) – An ASCII string of one or more PEM-encoded certs or a bytes object containing DER-encoded certificates. You can set the default per-connection by passing the *ssl_ca_data* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_CA_DATA*.

Return type *None*

tag (*tag*)

Get a tagged object.

Parameters **tag** (*str*) – The tag name to retrieve

Returns The object created with the given tag

Return type *LDAPObject*

Raises *TagError* – if the given tag is not defined

unbind (*force=False*)

Send an unbind request and close the socket.

Parameters **force** (*bool*) – Unbind and close the socket even if other objects still hold a reference to it.

Raises *ConnectionUnbound* – if the connection has already been unbound

validate_modify (*dn, modlist, current=None*)

Run all configured validators for the given modify operation

Parameters

- **dn** (*str*) – The DN of the object being modified
- **modlist** (*list [Mod]*) – The sequence of changes to be performed
- **current** (*LDAPObject*) – The current known state of the object

Return type *None*

Raises *LDAPValidationError* – if any validator fails the operation

validate_object (*obj, write=True*)

Run all configured validators for the given object.

Parameters

- **obj** (`LDAPObject`) – The object to validate
- **write** (`bool`) – True if this is for a write operation (e.g. an add)

Return type `None`

Raises `LDAPValidationError` – if any validator fails the object

who_am_i (`**ctrl_kwds`)

Perform the “Who Am I?” extended operation. This will confirm the identity that the connection is bound to.

Returns A string describing the bound identity. One common form is “dn:cn=foo,dc=example,dc=org” but this will vary by server configuration and bind type/parameters.

Return type `str`

Additional keyword arguments are handled as *Controls*.

class `laurelin.ldap.base.LDAPResponse`

Bases: `object`

Empty object for storing response control values

class `laurelin.ldap.base.LDAPURI (uri)`

Bases: `object`

Represents a parsed LDAP URI as specified in RFC4516

Supported extensions:

- “StartTLS”

Variables

- **scheme** (`str`) – urlparse standard
- **netloc** (`str`) – urlparse standard
- **host_uri** (`str`) – scheme://netloc for use with LDAPSockect
- **dn** (`str`) – Distinguished name
- **attrs** (`list[str]`) – list
- **scope** (`Scope`) – one of the `Scope` constants
- **filter** (`str`) – The filter string
- **starttls** (`bool`) – True if StartTLS was requested

`DEFAULT_ATTRS = ['*']`

`DEFAULT_FILTER = '(objectClass=*)'`

`DEFAULT_SCOPE = Scope.BASE`

`DEFAULT_STARTTLS = False`

search (`**kwds`)

Perform the search operation described by the parsed URI

First opens a new connection with connection reuse disabled, then performs the search, and unbinds the connection. Server must allow anonymous read.

Additional keyword arguments are passed through into `LDAP.search()`.

```
class laurelin.ldap.base.ResponseHandle (ldap_conn, mid)
    Bases: laurelin.ldap.base.LDAPResponse

    Base for return from methods with multiple response messages.

    abandon ()
        Request to abandon an operation in progress

class laurelin.ldap.base.SearchReferenceHandle (uris, obj_kwds)
    Bases: object

    Returned when the server returns a SearchResultReference

    fetch ()
        Perform the reference search and return an iterator over results

class laurelin.ldap.base.SearchResultHandle (ldap_conn, message_id, fetch_result_refs,
                                              follow_referrals, obj_kwds)
    Bases: laurelin.ldap.base.ResponseHandle
```

8.1.6 laurelin.ldap.config module

Provides support for establishing an LDAP connection and environment via config files and dicts

`laurelin.ldap.config.activate_extensions` (*config_dict*)
 Activate the specified extensions. The dict must be formatted as follows:

```
{'extensions': [
    <module name>,
]}
```

Parameters `config_dict` (*dict*) – See above.

Return type `None`

`laurelin.ldap.config.create_connection` (*config_dict*)
 Create a new connection from a config dict formatted as follows:

```
{'connection': {
    'start_tls': <bool>, # optional, default False
    'simple_bind': { # optional, default no bind; mutually exclusive with sasl_
↪bind
        'username': <string username or bind dn>,
        'password': <string password>
    },
    'sasl_bind': { # optional, default no bind, mutually exclusive with simple_
↪bind
        'mech': <standard mech name>,
        <mech prop>: <mech value>, # required props varies by mech
    },
    <constructor param>: <constructor value>,
},
'objects': [ # optional
    {'dn': <object dn>, # OR...
     'rdn': <dn relative to connection base object>,
     'tag': <unique tag name>,
     <object param>: <object value>,
```

(continues on next page)

(continued from previous page)

```

    },
    # ...
]
}

```

<constructor param> must be one of the [LDAP](#) constructor keyword arguments.

For validators you can optionally give the full path to the validator to use as a string, e.g. `['laurelin.ldap.schema.SchemaValidator']`

For `default_filter_syntax` give one of the strings “STANDARD” or “SIMPLE” (case-insensitive).

For objects (optional):

- If the `dn` parameter is specified, it is taken as an absolute DN.
- You can specify the `rdn` parameter instead to create the object as a child of the connection’s base object (the base of the tree).
- The `tag` parameter is required; this is how created objects are accessed ([LDAP.tag\(\)](#)).
- Additional <object param> will be passed as keywords to [LDAP.obj\(\)](#).
- If `relative_search_scope` is specified, use one of the strings *base*, *one*, or *sub*.
- The server will not be queried to create these objects, so they will have no local attributes. Call [LDAPObject.refresh\(\)](#) if you need to query attributes.

Note on binding: You can always manually call [LDAP.simple_bind\(\)](#) or [LDAP.sasl_bind\(\)](#) on the [LDAP](#) instance returned from this method if statically configuring bind credentials is not desirable.

Parameters `config_dict` – See above.

Returns The new LDAP instance with any objects created and tagged.

Raises `TypeError` – if any required object parameters are missing

`laurelin.ldap.config.load_config_dict` (*config_dict*)

Load config parameters from a dictionary. Must be formatted in the same was as `load_file`

Parameters `config_dict` (*dict*) – The config dictionary. See format in `load_file`.

Returns The LDAP connection if one was defined, None otherwise

Return type [LDAP](#) or None

`laurelin.ldap.config.load_file` (*path*, *file_decoder=None*)

Load a config file. Must decode to dict with all components described on other methods as optional sections/keys. A YAML example:

```

extensions:
- laurelin.extensions.descattrs
- laurelin.extensions.netgroups
global:
  SSL_CA_PATH: /etc/ldap/cacerts
  IGNORE_EMPTY_LIST: true
connection:
  server: ldap://dir01.example.org
  start_tls: true
  simple_bind:
    username: testuser
    passowrd: testpassword

```

(continues on next page)

(continued from previous page)

```

connect_timeout: 30
objects:
- rdn: ou=people
  tag: posix_user_base
- rdn: ou=groups
  tag: posix_group_base
- rdn: ou=netgroups
  tag: netgroup_base

```

Parameters

- **path** – A path to a config file. Provides support for YAML and JSON format, or you can specify your own decoder that returns a dict.
- **file_decoder** – A callable returning a dict when passed a file-like object

Returns The LDAP connection if one was defined, None otherwise

Return type *LDAP* or None

Raises *RuntimeError* – if an unsupported file extension was given without the `file_decoder` argument.

`laurelin.ldap.config.normalize_global_config_param(key)`

Normalize a global config key. Does not check validity of the key.

Parameters **key** (*str*) – User-supplied global config key

Returns The normalized key formatted as an attribute of *LDAP*

Return type *str*

`laurelin.ldap.config.set_global_config(global_config_dict)`

Set the global defaults. The dict must be formatted as follows:

```

{'global': {
    <config param>: <config value>,
  }
}

```

<config param> must match one of the `DEFAULT_` attributes on *LDAP*. The `DEFAULT_` prefix is optional and dict keys are case-insensitive. Any parameters not specified will keep the hard-coded default.

Parameters **global_config_dict** (*dict*) – See above.

Return type None

Raises *KeyError* – if the dict is incorrectly formatted or contains unknown config parameters

8.1.7 laurelin.ldap.constants module

Global constant classes.

class `laurelin.ldap.constants.DerefAliases`

Bases: *object*

DerefAliases constants. These instruct the server when to automatically resolve an alias object, rather than return the alias object itself

ALWAYS = DerefAliases.ALWAYS
dereferences both the search base object and results

BASE = DerefAliases.BASE
dereferences the search base object, but not search results

NEVER = DerefAliases.NEVER
always return the alias object

SEARCH = DerefAliases.SEARCH
dereferences search results, but not the base object itself

class `laurelin.ldap.constants.FilterSyntax`
Bases: `object`

Filter syntax selection constants. Used to determine which filter syntax to use when parsing a search filter.

SIMPLE = FilterSyntax.SIMPLE

STANDARD = FilterSyntax.STANDARD

UNIFIED = FilterSyntax.UNIFIED

static string (*str*)
Convert filter syntax string to constant

class `laurelin.ldap.constants.Scope`
Bases: `object`

Scope constants. These instruct the server how far to take a search, relative to the base object

BASE = Scope.BASE
Only search the base object

ONE = Scope.ONE
Search the base object and its immediate children

ONELEVEL = Scope.ONE

SUB = Scope.SUB
Search the base object and all of its descendants

SUBTREE = Scope.SUB

static constant (*c*)
translate constants to RFC4516 URL scope string

static string (*str*)
translate RFC4516 URL scope strings to constant

8.1.8 `laurelin.ldap.exceptions` module

exception `laurelin.ldap.exceptions.Abandon`
Bases: `Exception`

Can be raised to cleanly exit a context manager and abandon unread results

exception `laurelin.ldap.exceptions.ConnectionAlreadyBound`
Bases: `laurelin.ldap.exceptions.InvalidBindState`

Only raised by LDAP.*Bind methods if the connection is already bound when called

exception `laurelin.ldap.exceptions.ConnectionUnbound`

Bases: `laurelin.ldap.exceptions.InvalidBindState`

Raised when any server operation is attempted after a connection is unbound/closed

exception `laurelin.ldap.exceptions.InvalidBindState`

Bases: `laurelin.ldap.exceptions.LDAPError`

Base class for exceptions related to bind state

exception `laurelin.ldap.exceptions.InvalidSyntaxError`

Bases: `laurelin.ldap.exceptions.LDAPValidationError`

Raised when syntax validation fails

exception `laurelin.ldap.exceptions.LDAPConnectionError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Error occurred creating connection to the LDAP server

exception `laurelin.ldap.exceptions.LDAPError`

Bases: `Exception`

Base class for all exceptions raised by laurelin

exception `laurelin.ldap.exceptions.LDAPExtensionError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Error occurred setting up an extension module

exception `laurelin.ldap.exceptions.LDAPSASLError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Error occurred involving the SASL client

exception `laurelin.ldap.exceptions.LDAPSchemaError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Error relating to setting up the LDAP schema

exception `laurelin.ldap.exceptions.LDAPSupportError`

Bases: `laurelin.ldap.exceptions.LDAPError`

A feature is not supported by the server

exception `laurelin.ldap.exceptions.LDAPTransactionError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Raised by actions not included in a modify transaction

exception `laurelin.ldap.exceptions.LDAPUnicodeWarning`

Bases: `laurelin.ldap.exceptions.LDAPWarning`, `UnicodeWarning`

Warning category for unicode issues relating to LDAP

exception `laurelin.ldap.exceptions.LDAPUnsolicitedMessage` (*lm*, *exc_msg*)

Bases: `Exception`

Raised when a message with ID 0 is returned from the server

This may indicate an incompatibility between laurelin and your server distribution and thus is outside the normal exception inheritance chain.

exception `laurelin.ldap.exceptions.LDAPValidationError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Raised when validation fails

exception `laurelin.ldap.exceptions.LDAPWarning`

Bases: `Warning`

Generic LDAP warning category

exception `laurelin.ldap.exceptions.MultipleSearchResults`

Bases: `laurelin.ldap.exceptions.UnexpectedSearchResults`

Got multiple search results when exactly one was required

exception `laurelin.ldap.exceptions.NoSearchResults`

Bases: `laurelin.ldap.exceptions.UnexpectedSearchResults`

Got no search results when one or more was required

exception `laurelin.ldap.exceptions.ProhibitedCharacterError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Raised when a prohibited character is detected in RFC4518 string prep

exception `laurelin.ldap.exceptions.TagError`

Bases: `laurelin.ldap.exceptions.LDAPError`

Error with an object tag

exception `laurelin.ldap.exceptions.UnexpectedResponseType`

Bases: `laurelin.ldap.exceptions.LDAPError`

The response did not contain the expected protocol operation

exception `laurelin.ldap.exceptions.UnexpectedSearchResults`

Bases: `laurelin.ldap.exceptions.LDAPError`

Base class for unhandled search result situations

8.1.9 `laurelin.ldap.extensible` module

class `laurelin.ldap.extensible.Extensible`

Bases: `object`

classmethod `EXTEND (method_list)`

Install extension methods to the class

Parameters **method_list** – A list of callables

Raises `LDAPExtensionError` – if a name is already an attribute of the class

8.1.10 `laurelin.ldap.filter` module

Contains utilities for handling filters.

See RFC4515 String Representation of Search Filters

`laurelin.ldap.filter.escape (text)`

Escape special characters

`laurelin.ldap.filter.parse (filter_str)`

Parse unified filter syntax. Fully compatible with standard filters, laurelin simple filters, and any intermingling of the two.

`laurelin.ldap.filter.parse_simple_filter(simple_filter_str)`

Laurelin defines its own, simpler format for filter strings. It uses the RFC 4515 standard format for the various comparison expressions, but with SQL-style logic operations. (Fully standard RFC 4515 filters are fully supported and used by default)

`laurelin.ldap.filter.parse_standard_filter(filter_str)`

Parse an RFC 4515 filter string to an `rfc4511.Filter`

`laurelin.ldap.filter.rfc4511_filter_to_rfc4515_string(fil)`

Reverse `parse_standard_filter()`, mainly used for testing

Parameters `fil` (*Filter*) – An `rfc4511.Filter` object

Returns An RFC 4515 compatible filter string

8.1.11 laurelin.ldap.ldapobject module

class `laurelin.ldap.ldapobject.LDAPObject` (*dn*, *attrs_dict=None*, *ldap_conn=None*,
relative_search_scope=Scope.SUB,
rdn_attr=None)

Bases: `laurelin.ldap.attrsdict.AttrsDict`, `laurelin.ldap.extensible.Extensible`

Represents a single object with optional server affinity.

Many methods will raise an exception if used without a server connection. To instantiate an `LDAPObject` bound to a server connection, use `LDAP.obj()`.

Attributes and values are stored using the mapping interface inherited from `AttrsDict`, where dict keys are case-insensitive attribute names, and dict values are a list of attribute values.

Value lists are automatically wrapped in `AttrValueList`. This allows the use of any schema-defined matching and syntax rules for the attribute type in list operations.

Parameters

- **dn** (*str*) – The DN of the object
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict or None*) – The object's attributes
- **ldap_conn** (*LDAP or None*) – The optional LDAP connection to use
- **relative_search_scope** (*Scope*) – One of the `Scope` constants, this is the default scope used when using this object's `LDAPObject.search()` method. New objects created below this one will inherit this attribute by default. This attribute also defines the behavior of `LDAPObject.find()`.
- **rdn_attr** (*str or None*) – The default attribute name used in RDN's for descendants of this object. If specified, this allows you to only specify the value for methods that have an `rdn` argument. You can always specify a full `attr=value` for `rdn` arguments as well to override this behavior. New objects created below this one will inherit this attribute by default.

add_attrs (*attrs_dict, **ctrl_kwds*)

Add new attribute values to this object.

Parameters **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) –
The new attributes to add to the object

Return type `None`

Additional keywords are passed through into `LDAPObject.modify()`.

add_child (*rdn*, *attrs_dict*, ***kws*)

Create a new object below this one.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object
- **attrs_dict** (*dict*(*str*, *list*[*str* or *bytes*]) or *AttrsDict* or *None*) – The attributes for the object

Returns The new object

Return type *LDAPObject*

Additional keyword arguments are passed through into `LDAP.add()`

compare (*attr*, *value*)

Ask the server if this object has a matching attribute value. The comparison will take place following the schema-defined matching rules and syntax rules.

Parameters

- **attr** (*str*) – The attribute name
- **value** (*str*) – The assertion value

Returns A response object, `bool()` evaluating to the result of the comparison

Return type *CompareResponse*

Raises *RuntimeError* – if this object is not bound to an LDAP connection

delete (***ctrl_kws*)

Delete the entire object from the server, and render this instance useless.

Additional keywords are passed through into `LDAP.delete()`.

Return type *None*

Raises *RuntimeError* – if this object is not bound to an LDAP connection

delete_attrs (*attrs_dict*, ***ctrl_kws*)

Delete specific attribute values given in *attrs_dict*. Specifying a zero-length list for any attribute will delete all values for that attribute.

Parameters **attrs_dict** (*dict*(*str*, *list*[*str* or *bytes*]) or *AttrsDict*) –
The attributes to delete from the object

Return type *None*

Additional keywords are passed through into `LDAPObject.modify()`.

delete_child (*rdn*, ***ctrl_kws*)

Delete a child object below this one.

Parameters **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object

Returns The *LDAPResponse* from the delete operation

Return type *LDAPResponse*

Additional keyword arguments are treated as controls.

find (*rdn*, *attrs=None*, ***kws*)

Obtain a single object below this one with the most efficient means possible.

The strategy used is based on the `relative_search_scope` property of this object.

- If it is `Scope.BASE`, this method will always raise an `LDAPError`.
- If it is `Scope.ONE`, then the absolute DN for the child object will be constructed, and a `Scope.BASE` search will be performed to get the object.
- If it is `Scope.SUB`, then a subtree search will be performed below this object, using the RDN as a search filter.

Additional keywords are passed through into `LDAPObject.search()`.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if `rdn_attr` is defined for this object
- **attrs** (*list[str]*) – Optional. The list of attribute names to obtain.

Returns The LDAP object

Return type `LDAPObject`

Raises

- **LDAPError** – if this object's `relative_search_scope` is `Scope.BASE`.
- **NoSearchResults** – if no object could be found matching `rdn`.
- **MultipleSearchResults** – if more than one object was found.
- **RuntimeError** – if this object is not bound to an LDAP connection
- **ValueError** – if the `relative_search_scope` is set to an invalid value.

format_ldif ()

Format the object as an LDIF string.

Returns The object encoded as an LDIF.

Return type `str`

get_child (*rdn*, *attrs=None*, ***kws*)

Query the server for a child object.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if `rdn_attr` is defined for this object
- **attrs** (*list[str] or None*) – The list of attributes to query

Returns The object populated with data from the server

Return type `LDAPObject`

Raises **RuntimeError** – if this object is not bound to an LDAP connection

Additional keywords are passed through into `LDAP.search()` and `LDAPObject`

has_object_class (*object_class*)

A convenience method which checks if this object has a particular objectClass. May query the server for the objectClass attribute if it is not yet known.

Parameters **object_class** – The objectClass to check for.

Returns True if the objectClass is present, False otherwise

Return type `bool`

mod_dn (*new_rdn*, *clean_attr=True*, *new_parent=None*, ***ctrl_kws*)

Change the object DN, and possibly its location in the tree.

Parameters

- **new_rdn** (*str*) – The new RDN of the object
- **clean_attr** (*bool*) – Optional, default True. Remove the attribute associated with the RDN when changing it.
- **new_parent** (*str*) – Optional. The absolute DN of the object's new parent.

Return type *None***Raises** *RuntimeError* – if this object is not bound to an LDAP connectionAdditional keywords are passed through into *LDAP.mod_dn()*.**mod_transaction** ()

Begin a modify transaction on this object. Important: This IS NOT an RFC 5805 transaction.

Return type *ModTransactionObject***modify** (*modlist*, ***ctrl_kwds*)

Perform a series of modify operations on this object atomically.

Parameters **modlist** (*list* [*Mod*]) – A list of *Mod* instances, e.g. [*Mod*(*Mod*.ADD, 'someAttr', ['value1', 'value2'])]**Return type** *None***Raises** *RuntimeError* – if this object is not bound to an LDAP connectionAdditional keywords are passed through into *LDAP.modify()*.**move** (*new_dn*, *clean_attr=True*, ***ctrl_kwds*)

Specify the complete new absolute DN for this object.

Parameters

- **new_dn** (*str*) – The new absolute DN for the object
- **clean_attr** (*bool*) – Optional, default True. Remove the attribute associated with the RDN when changing it.

Return type *None*Additional keywords are passed through into *LDAPObject.mod_dn()*.**obj** (*rdn*, *attrs_dict=None*, *tag=None*, ***kwds*)

Create a new object below this one.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object
- **attrs_dict** (*dict* (*str*, *list* [*str* or *bytes*]) or *AttrsDict* or *None*) – The attributes for the object
- **tag** (*str* or *None*) – Optional tag for the object

Returns The new object**Return type** *LDAPObject***Raises** *LDAPError* – if a *tag* is specified but this object is not bound to an LDAP connectionAdditional keywords are passed through into *LDAP.obj()* or the *LDAPObject* constructor.**rdn** (*rdn*)

Return an absolute DN from an RDN or RDN value

Parameters `rdn` (*str*) – The RDN, or RDN value if `rdn_attr` is defined for this object

Returns The absolute DN

Return type *str*

refresh (*attrs=None*)

Query the server to update the attributes on this object.

Parameters `attrs` (*list[str]*) – Optional. A list of attribute names to query. If not specified, will query the server for all user attributes.

Return type *None*

Raises `RuntimeError` – if this object is not bound to an LDAP connection

refresh_all ()

Query the server to update all user and operational attributes on this object.

Return type *None*

Raises `RuntimeError` – if this object is not bound to an LDAP connection

refresh_missing (*attrs*)

Potentially query the server for any listed attributes that are not yet defined on this object. If no listed attributes aren't defined, the query will not be performed. If a subset of the list is undefined, only those attributes will be queried.

Parameters `attrs` (*list[str]*) – A list of attribute names to check, and possibly query for.

Return type *None*

rename (*new_rdn, clean_attr=True, **ctrl_kwds*)

Change the object's RDN without changing it's location in the tree.

Parameters

- `new_rdn` (*str*) – The new RDN of the object
- `clean_attr` (*bool*) – Optional, default True. Remove the attribute associated with the RDN when changing it.

Return type *None*

Additional keywords are passed through into `LDAPObject.mod_dn()`.

replace_attrs (*attrs_dict, **ctrl_kwds*)

Replace all values on the given attributes with the passed values.

Parameters `attrs_dict` (*dict(str, list[str or bytes]) or AttrsDict*) – The new attributes to set on the object

Return type *None*

Additional keywords are passed through into `LDAPObject.modify()`.

search (*filter=None, attrs=None, **kwds*)

Perform a search below this object.

Parameters

- `filter` (*str*) – Optional. The filter string to use to filter returned objects.
- `attrs` (*list[str]*) – Optional. The list of attribute names to retrieve.

Returns An iterator over `LDAPObject` and possibly `SearchReferenceHandle`. See `LDAP.search()` for more details.

Return type *SearchResultHandle*

Additional keywords are passed through into `LDAP.search()`.

validate()

Validate the object, assuming all attributes are present locally

validate_modify(modlist)

Validate a modification list.

Parameters `modlist` (*list* [*Mod*]) – The list of modify operations to validate.

class `laurelin.ldap.ldapobject.ModTransactionObject` (*ldap_object*)

Bases: `laurelin.ldap.ldapobject.LDAPObject`

Provides a transaction-like construct for building up a single modify operation. Users should use `LDAPObject.mod_transaction()` rather than instantiating this directly.

Inherits all modify methods from `LDAPObject`, allowing users to utilize the familiar interface for modifications, but overrides the base `modify` method so that changes are not immediately applied on the server.

The state of attributes is mutated within this transaction object with each higher-level modify call (e.g., `LDAPObject.add_attrs()`) allowing the state to be inspected. When `ModTransactionObject.commit()` is invoked, the built-up series of raw modify operations is sent to the server, and the state of the underlying `LDAPObject` is mutated.

Since this ultimately constructs only one modify operation per commit, the transaction is atomic.

You can also call `mod_transaction()` on a transaction object to create a “checkpoint”. The local state of the transaction will be copied into a new transaction object. To “roll back”, just delete the new object without committing.

Example:

```
from laurelin.ldap import LDAP

with LDAP() as ldap:
    obj = ldap.base.get_child('cn=someobject')
    print(obj.get_attr('memberUid'))
    # ['foo', 'bar']
    with obj.mod_transaction() as trans:
        trans.add_attrs({'memberUid': ['foobar']})
        print(trans.get_attr('memberUid'))
        # ['foo', 'bar', 'foobar']
        print(obj.get_attr('memberUid'))
        # ['foo', 'bar']

        trans.delete_attrs({'memberUid': ['bar']})
        print(trans.get_attr('memberUid'))
        # ['foo', 'foobar']
        print(obj.get_attr('memberUid'))
        # ['foo', 'bar']

    with trans.mod_transaction() as checkpoint:
        print(checkpoint.get_attr('memberUid'))
        # ['foo', 'foobar']
        print(trans.get_attr('memberUid'))
        # ['foo', 'foobar']
        print(obj.get_attr('memberUid'))
        # ['foo', 'bar']
```

(continues on next page)

(continued from previous page)

```

        checkpoint.delete_attrs({'memberUid': ['foo']})
        print(checkpoint.get_attr('memberUid'))
        # ['foobar']
        print(trans.get_attr('memberUid'))
        # ['foo', 'foobar']
        print(obj.get_attr('memberUid'))
        # ['foo', 'bar']

        # Note: no commit on checkpoint, meaning we will be rolled back to
        ↳ the pre-checkpoint state

        # Now in rolled-back (actually just unchanged) state
        print(trans.get_attr('memberUid'))
        # ['foo', 'foobar']
        print(obj.get_attr('memberUid'))
        # ['foo', 'bar']

        trans.commit()

        # Transaction was committed, we can now see changes reflected in the original
        ↳ object:
        print(obj.get_attr('memberUid'))
        # ['foo', 'foobar']

```

You can also raise *Abandon* from within a transaction context manager to cleanly abandon the transaction and exit the context manager.

add_child (*rdn*, *attrs_dict*, ***kws*)

Raises an error if used in a transaction. Transactions can only modify one object at a time.

Raises *LDAPTransactionError* – if this method is called.

commit ()

Send the modify operation to the server and update the original local *LDAPObject*.

Return type *None*

delete (***ctrl_kws*)

Raises an error if used in a transaction. Transactions can only modify one object at a time.

Raises *LDAPTransactionError* – if this method is called.

delete_child (*rdn*, ***ctrl_kws*)

Raises an error if used in a transaction. Transactions can only modify one object at a time.

Raises *LDAPTransactionError* – if this method is called

format_mod_ldif ()

Format the modify operation as an LDIF

Returns The LDIF string describing the modify operation to be performed

Return type *str*

mod_dn (*new_rdn*, *clean_attr=True*, *new_parent=None*, ***ctrl_kws*)

Raises an error if used in a transaction. Transactions can only modify one object at a time.

Raises *LDAPTransactionError* – if this method is called.

modify (*modlist*, ***kws*)

Process and validate a partial transaction, and mutate the transaction object's local attributes. Does not send anything to the server.

Parameters `modlist` (*list* [*Mod*]) – A partial list of modify operations to include in the transaction.

Return type `None`

Raises `TypeError` – if any extra keyword arguments are passed to this function.

8.1.12 laurelin.ldap.modify module

Contains utilities for performing object modification

`laurelin.ldap.modify.AddModlist` (*cur_attrs*, *new_attrs*)

Generate a modlist to add only new attribute values that are not known to exist

`laurelin.ldap.modify.DeleteModlist` (*cur_attrs*, *del_attrs*)

Generate a modlist to delete only attribute values that are known to exist

class `laurelin.ldap.modify.Mod` (*op*, *attr*, *vals*)

Bases: `object`

Describes a single modify operation

ADD = `Mod.ADD`

DELETE = `Mod.DELETE`

REPLACE = `Mod.REPLACE`

static `op_to_string` (*op*)

Convert one of the `Mod` constants to a string, e.g. “ADD”, “REPLACE”, “DELETE”.

static `string` (*op*)

Translte LDIF changetype strings to constant. e.g. “replace” -> `Mod.REPLACE`

`laurelin.ldap.modify.Modlist` (*op*, *attrs_dict*)

Generate a modlist from a dictionary

8.1.13 laurelin.ldap.net module

Provides protocol-level interface for low-level sockets

class `laurelin.ldap.net.LDAPSocket` (*host_uri*, *connect_timeout=5*, *ssl_verify=True*,
ssl_ca_file=None, *ssl_ca_path=None*,
ssl_ca_data=None)

Bases: `object`

Holds a connection to an LDAP server.

Parameters

- **host_uri** (*str*) – “scheme://netloc” to connect to
- **connect_timeout** (*int*) – Number of seconds to wait for connection to be accepted
- **ssl_verify** (*bool*) – Validate the certificate and hostname on an SSL/TLS connection
- **ssl_ca_file** (*str*) – Path to PEM-formatted concatenated CA certificates file
- **ssl_ca_path** (*str*) – Path to directory with CA certs under hashed file names. See https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_load_verify_locations.html for more information about the format of this directory.

- **ssl_ca_data** (*str* or *bytes*) – An ASCII string of one or more PEM-encoded certs or a bytes object containing DER-encoded certificates.

LDAP_SOCKET_PATHS = ['/var/run/ldapi', '/var/run/slapd/ldapi', '/var/run/slapd-*.sock']

OID_DISCONNECTION_NOTICE = '1.3.6.1.4.1.1466.20036'

RECV_BUFFER = 4096

check_hostname (*cert_cn*, *cert*)

SSL check_hostname according to RFC 4513 sec 3.1.3. Compares supplied values against *self.host* to determine the validity of the cert.

Parameters

- **cert_cn** (*str*) – The common name of the cert
- **cert** (*dict*) – A dictionary representing the rest of the cert. Checks key subjectAltNames for a list of (type, value) tuples, where type is 'DNS' or 'IP'. DNS supports leading wildcard.

Return type `None`

Raises `LDAPConnectionError` – if no supplied values match *self.host*

close ()

Close the low-level socket connection.

recv_messages (*want_message_id*)

Iterate all messages with *want_message_id* being sent by the server.

Parameters **want_message_id** (*int*) – The desired message ID.

Returns An iterator over `rfc4511.LDAPMessage`.

recv_one (*want_message_id*)

Get the next message with *want_message_id* being sent by the server

Parameters **want_message_id** (*int*) – The desired message ID.

Returns The LDAP message

Return type `rfc4511.LDAPMessage`

sasl_init (*mechs*, ***props*)

Initialize a `puresasl.client.SASLClient`

sasl_mech

Obtain the chosen mechanism

sasl_process_auth_challenge (*challenge*)

Process an auth challenge and return the correct response

sasl_qop

Obtain the chosen quality of protection

send_message (*op*, *obj*, *controls=None*)

Create and send an `LDAPMessage` given an operation name and a corresponding object.

Operation names must be defined as component names in `laurelin.ldap.rfc4511.ProtocolOp` and the object must be of the corresponding type.

Parameters

- **op** (*str*) – The protocol operation name

- **obj** (*object*) – The associated protocol object (see `rfc4511.ProtocolOp` for mapping).
- **controls** (*rfc4511.Controls* or *None*) – Any request controls for the message

Returns The message ID for this message

Return type `int`

start_tls (*verify=True, ca_file=None, ca_path=None, ca_data=None*)

Install TLS layer on this socket connection.

Parameters

- **verify** (*bool*) – Validate the certificate and hostname on an SSL/TLS connection
- **ca_file** (*str*) – Path to PEM-formatted concatenated CA certificates file
- **ca_path** (*str*) – Path to directory with CA certs under hashed file names. See https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_load_verify_locations.html for more information about the format of this directory.
- **ca_data** (*str* or *bytes*) – An ASCII string of one or more PEM-encoded certs or a bytes object containing DER-encoded certificates.

8.1.14 laurelin.ldap.objectclass module

class `laurelin.ldap.objectclass.DefaultObjectClass` (*name*)

Bases: `laurelin.ldap.objectclass.ObjectClass`

The default `ObjectClass` returned by `get_object_class()` when the requested object class is undefined.

Users should probably never instantiate this.

class `laurelin.ldap.objectclass.ExtensibleObjectClass` (*spec*)

Bases: `laurelin.ldap.objectclass.ObjectClass`

The *extensibleObject* auxiliary objectClass allows entries that belong to it to hold any user attribute.

allowed_attr (*name*)

Check if the given attribute type name is allowed.

Parameters **name** – The name of the attribute type to check

Returns True if the given attribute type is allowed.

Return type `bool`

class `laurelin.ldap.objectclass.ObjectClass` (*spec*)

Bases: `object`

Parses an LDAP object class specification and implements superclass inheritance.

Each instantiation registers the names and OID specified so that they can later be access with `get_object_class()`.

See the `laurelin.ldap.schema` module source for example usages.

Parameters **spec** (*str*) – The LDAP specification for an object class

Raises `LDAPSchemaError` –

- if the schema is syntactically invalid
- if the OID specified has already been registered

- if one of the names specified has already been registered

Variables

- **oid** (*str*) – The specified OID
- **names** (*tuple* (*str*)) – All specified names
- **superclasses** (*list* [*str*]) – The list of all specified superclass names/OIDs.
- **kind** (*str*) – One of *ABSTRACT*, *STRUCTURAL*, or *AUXILIARY*
- **obsolete** (*bool*) – True if the objectClass has been marked obsolete.
- **my_must** (*list* [*str*]) – The list of required attribute types for this class
- **my_may** (*list* [*str*]) – The list of allowed attribute types for this class

allowed_attr (*name*)

Check if the given attribute type name is allowed.

Parameters **name** – The name of the attribute type to check

Returns True if the given attribute type is allowed.

Return type *bool*

may

Obtains all allowed attribute types after ascending the superclass specifications

must

Obtains all required attribute types after ascending the superclass specifications

required_attr (*name*)

Check if the given attribute type name is required.

Parameters **name** – The name of the attribute type to check

Returns True if the given attribute type is required.

Return type *bool*

`laurelin.ldap.objectclass.get_object_class` (*ident*)

Get an instance of *ObjectClass* associated with either a name or an OID

Parameters **ident** (*str*) – Either the numeric OID of the desired object class spec or one of its specified names

Returns The *ObjectClass* associated with the name/OID

Return type *ObjectClass*

8.1.15 laurelin.ldap.rules module

Base classes for syntax rules and matching rules

class `laurelin.ldap.rules.EqualityMatchingRule`

Bases: *laurelin.ldap.rules.MatchingRule*

Base class for all EQUALITY matching rules

do_match (*attribute_value*, *assertion_value*)

Perform equality matching

class `laurelin.ldap.rules.MatchingRule`

Bases: `object`

Base class for all matching rules

NAME = `''`,

Globally unique name for the matching rule. Most attribute type specs will reference rules using the name, but they can also use the OID. This must be defined by subclasses.

OID = `''`

Globally unique numeric OID for the matching rule. This must be defined by subclasses.

SYNTAX = `''`

The numeric OID for the syntax rule that assertion values must comply with. Subclasses must define this.

do_match (*attribute_value*, *assertion_value*)

Perform the match operation

match (*attribute_value*, *assertion_value*)

Prepare values and perform the match operation. Assumes values have already been validated.

prep_methods = `()`

A tuple of callables used to prepare attribute and asserion values. Subclasses may optionally define this.

prepare (*value*)

Prepare a string for matching

validate (*value*)

Perform validation according to the matching rule's syntax

class `laurelin.ldap.rules.MetaMatchingRule`

Bases: `type`

Metaclass registering OIDs and NAMES on subclasses

class `laurelin.ldap.rules.MetaSyntaxRule`

Bases: `type`

Metaclass registering OIDs on subclasses

class `laurelin.ldap.rules.RegexSyntaxRule`

Bases: `laurelin.ldap.rules.SyntaxRule`

For validating rules based on a regular expression. Most syntax rules can inherit from this.

regex = `''`

The regular expression defining the rule. Subclasses must define this attribute.

validate (*s*)

Validate a string against the regular expression.

Parameters *s* – Candidate string

Returns The regex match object

Return type MatchObject

Raises `InvalidSyntaxError` – if the string does not match

class `laurelin.ldap.rules.SyntaxRule`

Bases: `object`

Base class for all syntax rules

DESC = `''`

Short text description of the rule. Must be defined by subclasses.

OID = ''

The globally unique numeric OID of the syntax rule. Referenced in attribute type and matching rule specs. Must be defined by subclasses.

validate(*s*)

Validate a string. Must be implemented by subclasses.

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

`laurelin.ldap.rules.get_matching_rule(ident)`

Obtains matching rule instance for name or OID

`laurelin.ldap.rules.get_syntax_rule(oid)`

8.1.16 laurelin.ldap.schema module

Schema specifications from various RFCs

class `laurelin.ldap.schema.AttributeTypeDescription`

Bases: `laurelin.ldap.rules.RegexSyntaxRule`

DESC = 'Attribute Type Description'

OID = '1.3.6.1.4.1.1466.115.121.1.3'

regex = '^\\(* (?P<oid>[0-9]+(?:\\. [0-9]+)+)) (? : +NAME + (?P<name> (?: ' [A-Za-z] [A-Za-z0-

class `laurelin.ldap.schema.Binary`

Bases: `laurelin.ldap.rules.SyntaxRule`

DESC = 'Binary'

OID = '1.3.6.1.4.1.1466.115.121.1.5'

validate(*s*)

Validate a string. Must be implemented by subclasses.

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

class `laurelin.ldap.schema.BitString`

Bases: `laurelin.ldap.rules.RegexSyntaxRule`

DESC = 'Bit String'

OID = '1.3.6.1.4.1.1466.115.121.1.6'

regex = '^' [01]* 'B\$'

class `laurelin.ldap.schema.Boolean`

Bases: `laurelin.ldap.rules.SyntaxRule`

DESC = 'Boolean'

OID = '1.3.6.1.4.1.1466.115.121.1.7'

validate(*s*)

Validate a string. Must be implemented by subclasses.

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

```
class laurelin.ldap.schema.Certificate
    Bases: laurelin.ldap.rules.SyntaxRule
```

```
    DESC = 'Certificate'
```

```
    OID = '1.3.6.1.4.1.1466.115.121.1.8'
```

```
    validate(s)
```

```
        Validate a string. Must be implemented by subclasses.
```

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

```
class laurelin.ldap.schema.CountryString
    Bases: laurelin.ldap.rules.RegexSyntaxRule
```

```
    DESC = 'Country String'
```

```
    OID = '1.3.6.1.4.1.1466.115.121.1.11'
```

```
    regex = "^[A-Za-z0-9'()+,./=:? -]{2}$"
```

```
class laurelin.ldap.schema.DITContentRuleDescription
    Bases: laurelin.ldap.rules.RegexSyntaxRule
```

```
    DESC = 'DIT Content Rule Description'
```

```
    OID = '1.3.6.1.4.1.1466.115.121.1.16'
```

```
    regex = "^\\( * ( ? P < oid > [ 0 - 9 ] + ( ? : \\ . [ 0 - 9 ] + ) + ) ( ? : + NAME + ( ? : ' [ A - Z a - z ] [ A - Z a - z 0 - 9 - ] * ' | \\ (
```

```
class laurelin.ldap.schema.DITStructureRuleDescription
    Bases: laurelin.ldap.rules.RegexSyntaxRule
```

```
    DESC = 'DIT Structure Rule Description'
```

```
    OID = '1.3.6.1.4.1.1466.115.121.1.17'
```

```
    regex = "^\\( * ( ? : [ 0 - 9 ] + ) ( ? : + NAME + ( ? : ' [ A - Z a - z ] [ A - Z a - z 0 - 9 - ] * ' | \\ ( * ' [ A - Z a - z ] [ A - Z a - z 0 - 9 - ] * ' | \\ (
```

```
class laurelin.ldap.schema.DeliveryMethod
    Bases: laurelin.ldap.rules.RegexSyntaxRule
```

```
    DESC = 'Delivery Method'
```

```
    OID = '1.3.6.1.4.1.1466.115.121.1.14'
```

```
    regex = '^ ( ? : any | mhs | physical | telex | teletext | g3fax | g4fax | ia5 | videotext | telephone ) ( \\ s *
```

```
class laurelin.ldap.schema.DirectoryString
    Bases: laurelin.ldap.rules.SyntaxRule
```

```
    DESC = 'Directory String'
```

```
    OID = '1.3.6.1.4.1.1466.115.121.1.15'
```

```
    validate(s)
```

```
        Validate a string. Must be implemented by subclasses.
```

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

```
class laurelin.ldap.schema.DistinguishedName
```

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'DN'

OID = '1.3.6.1.4.1.1466.115.121.1.12'

regex = '^(?:[A-Za-z][A-Za-z0-9-]*|[0-9]+(?:\.[0-9]+)+)=(?: (?: [^",; <>\\0\\\\ \#]=) | \\\\"'

```
class laurelin.ldap.schema.EnhancedGuide
```

Bases: *laurelin.ldap.rules.SyntaxRule*

DESC = 'Enhanced Guide'

OID = '1.3.6.1.4.1.1466.115.121.1.21'

validate(s)

Validate a string. Must be implemented by subclasses.

Parameters s – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

```
class laurelin.ldap.schema.FacsimileTelephoneNumber
```

Bases: *laurelin.ldap.rules.SyntaxRule*

DESC = 'Facsimile Telephone Number'

OID = '1.3.6.1.4.1.1466.115.121.1.22'

validate(s)

Validate a string. Must be implemented by subclasses.

Parameters s – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

```
class laurelin.ldap.schema.Fax
```

Bases: *laurelin.ldap.rules.SyntaxRule*

DESC = 'Fax'

OID = '1.3.6.1.4.1.1466.115.121.1.23'

validate(s)

Validate a string. Must be implemented by subclasses.

Parameters s – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

```
class laurelin.ldap.schema.GeneralizedTime
```

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Generalized Time'

OID = '1.3.6.1.4.1.1466.115.121.1.24'

regex = '^([0-9]{4})([0-9]{2})([0-9]{2})([0-9]{2})([0-9]{2})([0-9]{2})?([0-9]{2})?([.,][0-9]+)?(\\\"'

validate (*s*)

Validate a string against the regular expression.

Parameters *s* – Candidate string

Returns The regex match object

Return type MatchObject

Raises *InvalidSyntaxError* – if the string does not match

class laurelin.ldap.schema.**Guide**

Bases: *laurelin.ldap.schema.EnhancedGuide*

DESC = 'Guide'

OID = '1.3.6.1.4.1.1466.115.121.1.25'

validate (*s*)

Validate a string. Must be implemented by subclasses.

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

class laurelin.ldap.schema.**IA5String**

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'IA5 String'

OID = '1.3.6.1.4.1.1466.115.121.1.26'

regex = '^[\x00-\x7f]*\$'

class laurelin.ldap.schema.**Integer**

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'INTEGER'

OID = '1.3.6.1.4.1.1466.115.121.1.27'

regex = '^-[1-9][0-9]*\$'

class laurelin.ldap.schema.**JPEG**

Bases: *laurelin.ldap.rules.SyntaxRule*

DESC = 'JPEG'

OID = '1.3.6.1.4.1.1466.115.121.1.28'

validate (*s*)

Validate a string. Must be implemented by subclasses.

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

class laurelin.ldap.schema.**LDAPSyntaxDescription**

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'LDAP Syntax Description'

OID = '1.3.6.1.4.1.1466.115.121.1.54'

regex = '^\\(* (?P<oid>[0-9]+(?:\\. [0-9]+)+) (? : +DESC + ' (? : \\ \\ 5 [Cc] | \\ \\ 27 | [^' \\ \\])


```

class laurelin.ldap.schema.MatchingRuleDescription
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'Matching Rule Description'

    OID = '1.3.6.1.4.1.1466.115.121.1.30'

    regex = '^\\( *(?P<oid>[0-9]+(?:\\. [0-9]+)+) (?: +NAME +(?: '[A-Za-z] [A-Za-z0-9-]*' | \\(
class laurelin.ldap.schema.MatchingRuleUseDescription
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'Matching Rule Use Description'

    OID = '1.3.6.1.4.1.1466.115.121.1.31'

    regex = '^\\( *(?P<oid>[0-9]+(?:\\. [0-9]+)+) (?: +NAME +(?: '[A-Za-z] [A-Za-z0-9-]*' | \\(
class laurelin.ldap.schema.NameAndOptionalUID
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'Name And Optional UID'

    OID = '1.3.6.1.4.1.1466.115.121.1.34'

    regex = '^(?:[A-Za-z] [A-Za-z0-9-]*|[0-9]+(?:\\. [0-9]+)+)=(?: (?: [^",; <>\\\\0\\\\\\\\ #=] | \\\\
class laurelin.ldap.schema.NameFormDescription
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'Name Form Description'

    OID = '1.3.6.1.4.1.1466.115.121.1.35'

    regex = '^\\( *(?:[0-9]+(?:\\. [0-9]+)+) (?: +NAME +(?: '[A-Za-z] [A-Za-z0-9-]*' | \\( *'[A
class laurelin.ldap.schema.NumericString
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'Numeric String'

    OID = '1.3.6.1.4.1.1466.115.121.1.36'

    regex = '^[0-9 ]+$'
class laurelin.ldap.schema.OID
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'OID'

    OID = '1.3.6.1.4.1.1466.115.121.1.38'

    regex = '^(?:[A-Za-z] [A-Za-z0-9-]*|[0-9]+(?:\\. [0-9]+)+)$'
class laurelin.ldap.schema.ObjectClassDescription
    Bases: laurelin.ldap.rules.RegexSyntaxRule

    DESC = 'Object Class Description'

    OID = '1.3.6.1.4.1.1466.115.121.1.37'

    regex = '^\\( *(?P<oid>[0-9]+(?:\\. [0-9]+)+) (?: +NAME +(?P<name> (?: '[A-Za-z] [A-Za-z0-
class laurelin.ldap.schema.OctetString
    Bases: laurelin.ldap.rules.SyntaxRule

    DESC = 'Octet String'

    OID = '1.3.6.1.4.1.1466.115.121.1.40'

```

validate(s)

Validate a string. Must be implemented by subclasses.

Parameters s – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

class laurelin.ldap.schema.OtherMailbox

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Other Mailbox'

OID = '1.3.6.1.4.1.1466.115.121.1.39'

regex = "^([A-Za-z0-9'()+,./:~ -]+\\\$[\\x00-\\x7f]*\$)"

class laurelin.ldap.schema.PostalAddress

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Postal Address'

OID = '1.3.6.1.4.1.1466.115.121.1.41'

regex = '^((\\\\\\\\24|\\\\\\\\5[cC]|[^\$\\\\\\\\])+(\\\\\$ (\\\\\\\\24|\\\\\\\\5[cC]|[^\$\\\\\\\\])+)*)\$'

class laurelin.ldap.schema.PrintableString

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Printable String'

OID = '1.3.6.1.4.1.1466.115.121.1.44'

regex = "^([A-Za-z0-9'()+,./:~ -]+)\$"

class laurelin.ldap.schema.SchemaValidator

Bases: *laurelin.ldap.validation.Validator*

Ensures parameters conform to the available defined schema

validate_object(obj, write=True)

Validates an object when all attributes are present

- Requires the objectClass attribute
- Checks that all attributes required by the objectClass are defined
- Checks that all attributes are allowed by the objectClass
- Performs validation against the attribute type spec for all attributes

class laurelin.ldap.schema.SubstringAssertion

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Substring Assertion'

OID = '1.3.6.1.4.1.1466.115.121.1.58'

regex = '(?: (\\\\\\\\2[aA]|\\\\\\\\5[cC]|[^*\\\\\\\\])+) ?* (?: (\\\\\\\\2[aA]|\\\\\\\\5[cC]|[^*\\\\\\\\])+) ?*'

class laurelin.ldap.schema.TelephoneNumber

Bases: *laurelin.ldap.rules.SyntaxRule*

DESC = 'Telephone Number'

OID = '1.3.6.1.4.1.1466.115.121.1.50'

validate(*s*)

Validate a string. Must be implemented by subclasses.

Parameters *s* – Candidate string

Returns Any useful value for the rule

Raises *InvalidSyntaxError* – if the string is invalid

class laurelin.ldap.schema.**TeletextTerminalIdentifier**

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Teletex Terminal Identifier'

OID = '1.3.6.1.4.1.1466.115.121.1.51'

regex = '^([A-Za-z0-9'()+,./=:? -]+(?:\\\$?:graphic|control|misc|page|private):(?:[\\x

class laurelin.ldap.schema.**TelexNumber**

Bases: *laurelin.ldap.rules.RegexSyntaxRule*

DESC = 'Telex Number'

OID = '1.3.6.1.4.1.1466.115.121.1.52'

regex = '^([A-Za-z0-9'()+,./=:? -]+\\\$[A-Za-z0-9'()+,./=:? -]+\\\$[A-Za-z0-9'()+,./=:?

class laurelin.ldap.schema.**bitStringMatch**

Bases: *laurelin.ldap.rules.EqualityMatchingRule*

NAME = ('bitStringMatch',)

OID = '2.5.13.16'

SYNTAX = '1.3.6.1.4.1.1466.115.121.1.6'

class laurelin.ldap.schema.**booleanMatch**

Bases: *laurelin.ldap.rules.EqualityMatchingRule*

NAME = ('booleanMatch',)

OID = '2.5.13.13'

SYNTAX = '1.3.6.1.4.1.1466.115.121.1.7'

class laurelin.ldap.schema.**caseExactIA5Match**

Bases: *laurelin.ldap.rules.EqualityMatchingRule*

NAME = ('caseExactIA5Match',)

OID = '1.3.6.1.4.1.1466.109.114.1'

SYNTAX = '1.3.6.1.4.1.1466.115.121.1.26'

prep_methods = (<function Transcode>, <function Map.characters>, <function Normalize>,

class laurelin.ldap.schema.**caseExactMatch**

Bases: *laurelin.ldap.rules.EqualityMatchingRule*

NAME = ('caseExactMatch',)

OID = '2.5.13.5'

SYNTAX = '1.3.6.1.4.1.1466.115.121.1.15'

prep_methods = (<function Transcode>, <function Map.characters>, <function Normalize>,

class laurelin.ldap.schema.**caseIgnoreIA5Match**

Bases: *laurelin.ldap.rules.EqualityMatchingRule*

```
NAME = ('caseIgnoreIA5Match',)
OID = '1.3.6.1.4.1.1466.109.114.2'
SYNTAX = '1.3.6.1.4.1.1466.115.121.1.26'
prep_methods = (<function Transcode>, <function Map.all>, <function Normalize>, <function Transcode>)

class laurelin.ldap.schema.caseIgnoreListMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('caseIgnoreListMatch',)
    OID = '2.5.13.11'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.41'
    prep_methods = (<function Transcode>, <function Map.all>, <function Normalize>, <function Transcode>)

class laurelin.ldap.schema.caseIgnoreMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('caseIgnoreMatch',)
    OID = '2.5.13.2'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.15'
    prep_methods = (<function Transcode>, <function Map.all>, <function Normalize>, <function Transcode>)

class laurelin.ldap.schema.directoryStringFirstComponentMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('directoryStringFirstComponentMatch',)
    OID = '2.5.13.31'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.15'

class laurelin.ldap.schema.distinguishedNameMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('distinguishedNameMatch',)
    OID = '2.5.13.1'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.12'
    do_match(attribute_value, assertion_value)
        Perform equality matching

class laurelin.ldap.schema.generalizedTimeMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('generalizedTimeMatch',)
    OID = '2.5.13.27'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.24'
    do_match(attribute_value, assertion_value)
        Perform equality matching

class laurelin.ldap.schema.integerFirstComponentMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('integerFirstComponentMatch',)
    OID = '2.5.13.29'
```

```

    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.27'
class laurelin.ldap.schema.integerMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('integerMatch',)
    OID = '2.5.13.14'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.27'
class laurelin.ldap.schema.numericStringMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('numericStringMatch',)
    OID = '2.5.13.8'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.36'
    prep_methods = (<function Transcode>, <function Map.characters>, <function Normalize>,
class laurelin.ldap.schema.objectIdentifierFirstComponentMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('objectIdentifierFirstComponentMatch',)
    OID = '2.5.13.30'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.38'
class laurelin.ldap.schema.objectIdentifierMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('objectIdentifierMatch',)
    OID = '2.5.13.0'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.38'
class laurelin.ldap.schema.octetStringMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('octetStringMatch',)
    OID = '2.5.13.17'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.40'
class laurelin.ldap.schema.telephoneNumberMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('telephoneNumberMatch',)
    OID = '2.5.13.20'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.50'
    prep_methods = (<function Transcode>, <function Map.all>, <function Normalize>, <funct.
class laurelin.ldap.schema.uniqueMemberMatch
    Bases: laurelin.ldap.rules.EqualityMatchingRule
    NAME = ('uniqueMemberMatch',)
    OID = '2.5.13.23'
    SYNTAX = '1.3.6.1.4.1.1466.115.121.1.34'

```

8.1.17 laurelin.ldap.validation module

class `laurelin.ldap.validation.DisabledValidationContext` (*ldap*, *disabled_validators=None*)

Bases: `object`

This should be created by calling `LDAP.disable_validation()` and never directly instantiated.

class `laurelin.ldap.validation.Validator`

Bases: `object`

Abstract base class for a validator. All validators must inherit from here and ensure the public interface is fully implemented.

validate_modify (*dn*, *modlist*, *current*)

Validate a modify operation.

By default, validate all attributes for writing.

Parameters

- **dn** (*str*) – The DN of the object being modified
- **modlist** (*list* [*Mod*]) – The list of modify operations to be performed this transaction
- **current** (*LDAPObject* or *None*) – The known state of the object prior to modification

Returns `None`

Raises `LDAPValidationError` – if any modify operation is invalid

validate_object (*obj*, *write=True*)

Validate an object when all attributes are present.

By default, validate all attributes on the object.

Parameters

- **obj** (*LDAPObject*) – An LDAP object with all attributes defined
- **write** (*bool*) – True if we are validating a write operation to the database

Returns `None`

Raises `LDAPValidationError` – if the object is invalid in any way

8.1.18 Module contents

`laurelin.ldap`

Imports base objects for user import and defines user utility functions

class `laurelin.ldap.LDAP` (*server=None*, *base_dn=None*, *reuse_connection=None*, *connect_timeout=None*, *search_timeout=None*, *deref_aliases=None*, *strict_modify=None*, *ssl_verify=None*, *ssl_ca_file=None*, *ssl_ca_path=None*, *ssl_ca_data=None*, *fetch_result_refs=None*, *default_sasl_mech=None*, *sasl_fatal_downgrade_check=None*, *default_criticality=None*, *follow_referrals=None*, *validators=None*, *warn_empty_list=None*, *error_empty_list=None*, *ignore_empty_list=None*, *filter_syntax=None*)

Bases: `laurelin.ldap.extensible.Extensible`

Provides the connection to the LDAP DB. All constructor parameters have a matching global default as a class property on `LDAP`

Parameters

- **server** (*str* or `LDAPSock`) – URI string to connect to or an `LDAPSock` to reuse
- **base_dn** (*str*) – The DN of the base object
- **reuse_connection** (*bool*) – Allows the socket connection to be reused and reuse an existing socket if possible.
- **connect_timeout** (*int*) – Number of seconds to wait for connection to be accepted.
- **search_timeout** (*int*) – Number of seconds to wait for a search to complete. Partial results will be returned when the timeout is reached. Can be overridden on a per-search basis by setting the `search_timeout` keyword on `LDAP.search()`.
- **deref_aliases** (`DerefAliases`) – One of the `DerefAliases` constants. Instructs the server how to handle alias objects in search results. Can be overridden on a per-search basis by setting the `deref_aliases` keyword on `LDAP.search()`.
- **strict_modify** (*bool*) – Use the strict modify strategy. If set to True, guarantees that another search will not take place before a modify operation. May potentially produce more server errors.
- **ssl_verify** (*bool*) – Validate the certificate and hostname on an SSL/TLS connection
- **ssl_ca_file** (*str*) – Path to PEM-formatted concatenated CA certificates file
- **ssl_ca_path** (*str*) – Path to directory with CA certs under hashed file names. See https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_load_verify_locations.html for more information about the format of this directory.
- **ssl_ca_data** (*str* or *bytes*) – An ASCII string of one or more PEM-encoded certs or a bytes object containing DER-encoded certificates.
- **fetch_result_refs** (*bool*) – Fetch `searchResultRef` responses in search results. Can be overridden on a per-search basis by setting the `fetch_result_refs` keyword on `LDAP.search()`.
- **default_sasl_mech** (*str*) – Name of the default SASL mechanism. Bind will fail if the server does not support the mechanism. (Examples: DIGEST-MD5, GSSAPI)
- **sasl_fatal_downgrade_check** (*bool*) – Set to False to make potential downgrade attack check non-fatal.
- **default_criticality** (*bool*) – Set to True to make controls critical by default, set to False to make non-critical
- **follow_referrals** (*bool*) – Automatically follow referral results
- **validators** (*list* [`Validator`]) – A list of `Validator` instances to apply to this connection.
- **warn_empty_list** (*bool*) – Default False. Set to True to emit a warning when an empty value list is passed to `LDAP.modify()`, `LDAP.replace_attrs()`, or `LDAP.delete_attrs()` or their `LDAPObject` counterparts.
- **error_empty_list** (*bool*) – Default False. Set to True to raise an exception when an empty value list is passed to `LDAP.modify()`, `LDAP.replace_attrs()`, or `LDAP.delete_attrs()` or their `LDAPObject` counterparts.

- **ignore_empty_list** (*bool*) – Default False. Set to True to ignore empty value lists passed to `LDAP.modify()`, `LDAP.replace_attrs()`, or `LDAP.delete_attrs()` or their LDAPObject counterparts. This will be default True in a future release.
- **filter_syntax** (*FilterSyntax*) – The default search filter syntax selection. Must be one of the *FilterSyntax* constants. Can be overridden on a per-search basis by setting the `filter_syntax` keyword on `LDAP.search()`. Defaults to *FilterSyntax.STANDARD* for RFC4515-compliant filter string syntax.

The class can be used as a context manager, which will automatically unbind and close the connection when the context manager exits.

Example:

```
with LDAP() as ldap:
    raise Exception()
# ldap is closed and unbound

with LDAP() as ldap:
    print('hello')
# ldap is closed and unbound
```

```
DEFAULT_BASE_DN = None
DEFAULT_CONNECT_TIMEOUT = 5
DEFAULT_CRITICALITY = False
DEFAULT_DEREF_ALIASES = DerefAliases.ALWAYS
DEFAULT_ERROR_EMPTY_LIST = False
DEFAULT_FETCH_RESULT_REFS = True
DEFAULT_FILTER = '(objectClass=*)'
DEFAULT_FILTER_SYNTAX = FilterSyntax.UNIFIED
DEFAULT_FOLLOW_REFERRALS = True
DEFAULT_IGNORE_EMPTY_LIST = False
DEFAULT_REUSE_CONNECTION = True
DEFAULT_SASL_FATAL_DOWNGRADE_CHECK = True
DEFAULT_SASL_MECH = None
DEFAULT_SEARCH_TIMEOUT = 0
DEFAULT_SERVER = 'ldap://localhost'
DEFAULT_SSL_CA_DATA = None
DEFAULT_SSL_CA_FILE = None
DEFAULT_SSL_CA_PATH = None
DEFAULT_SSL_VERIFY = True
DEFAULT_STRICT_MODIFY = False
DEFAULT_VALIDATORS = None
DEFAULT_WARN_EMPTY_LIST = False
```


DELETE_ALL = <delete all values>

Use with modify replace/delete in place of an attribute list to delete all values for the attribute

LOG_FORMAT = '[%(asctime)s] %(name)s %(levelname)s : %(message)s'

NO_ATTRS = '1.1'

OID_OBJ_CLASS_ATTR = '1.3.6.1.4.1.4203.1.5.2'

OID_STARTTLS = '1.3.6.1.4.1.1466.20037'

OID_WHOAMI = '1.3.6.1.4.1.4203.1.11.3'

static activate_extension (*module_name*)

Import the module name and call the `activate_extension` function on the module.

Parameters **module_name** (*str*) – The name of the module to import and activate

Returns The imported module

Return type module

add (*dn*, *attrs_dict*, ***kws*)

Add new object and return corresponding LDAPObject on success.

Parameters

- **dn** (*str*) – The new object's DN
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The new attributes for the object

Returns The new object

Return type *LDAPObject*

Raises

- *ConnectionUnbound* – if the connection has been unbound
- *TypeError* – if arguments are of invalid type
- *LDAPValidationError* – if the object fails any configured validator
- *LDAPError* – if we get a non-success result

Additional keyword arguments are handled as *Controls* and then passed through into `LDAP.obj()`.

add_attrs (*dn*, *attrs_dict*, *current=None*, ***ctrl_kws*)

Add new attribute values to existing object.

Parameters

- **dn** (*str*) – The DN of the object to modify
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The new attributes to add to the object
- **current** (*LDAPObject or None*) – The current known state of the object. Used for ensuring we don't send duplicate attributes to the server and for validation.

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

add_if_not_exists (*dn*, *attrs_dict*)

Add object if it doesn't exist

- Gets and returns the object at DN if it exists, otherwise create the object using the attrs dictionary
- Always returns an `LDAPObject` corresponding to the final state of the DB

Parameters

- **dn** (*str*) – The object DN
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The attributes to use if adding the object

Returns The new or existing object

Return type *LDAPObject*

add_or_mod_add_if_exists (*dn, attrs_dict*)

Add object if it doesn't exist, otherwise add_attrs

- If the object at DN exists, perform an add modification using the attrs dictionary. Otherwise, create the object using the attrs dictionary.
- This ensures that, for the attributes mentioned in attrs, AT LEAST those values will exist on the given DN, regardless of prior state of the DB.
- Always returns an *LDAPObject* corresponding to the final state of the DB

Parameters

- **dn** (*str*) – The object DN
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The objects minimum attributes

Returns The new or modified object

Return type *LDAPObject*

add_or_mod_replace_if_exists (*dn, attrs_dict*)

Add object if it doesn't exist, otherwise replace_attrs

- If the object at DN exists, perform a replace modification using the attrs dictionary. Otherwise, create the object using the attrs dictionary.
- This ensures that, for the attributes mentioned in attrs, ONLY those values will exist on the given DN regardless of prior state of the DB.
- Always returns an *LDAPObject* corresponding to the final state of the DB

Parameters

- **dn** (*str*) – The object DN
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The objects new required attributes

Returns The new or modified object

Return type *LDAPObject*

close (*force=False*)

Send an unbind request and close the socket.

Parameters **force** (*bool*) – Unbind and close the socket even if other objects still hold a reference to it.

Raises *ConnectionUnbound* – if the connection has already been unbound

compare (*dn*, *attr*, *value*, ***ctrl_kwds*)

Ask the server if a particular DN has a matching attribute value. The comparison will take place following the schema-defined matching rules and syntax rules.

Parameters

- **dn** (*str*) – The DN of the object
- **attr** (*str*) – The attribute name
- **value** (*str*) – The assertion value

Returns A response object, `bool()` evaluating to the result of the comparison

Return type *CompareResponse*

Raises

- *ConnectionUnbound* – if the connection has been unbound
- *LDAPError* – if we got a result other than `compareTrue` or `compareFalse`

Additional keyword arguments are handled as *Controls*.

static default_warnings ()

Always take the default action for warnings

delete (*dn*, ***ctrl_kwds*)

Delete an object.

Parameters **dn** (*str*) – The DN of the object to delete

Returns A response object

Return type *LDAPResponse*

Raises *ConnectionUnbound* – if the connection has been unbound

Additional keyword arguments are handled as *Controls*.

delete_attrs (*dn*, *attrs_dict*, *current=None*, ***ctrl_kwds*)

Delete specific attribute values from *attrs_dict*.

Specifying a 0-length entry will delete all values.

Parameters

- **dn** (*str*) – The DN of the object to modify
- **attrs_dict** (*dict(str, list[str or bytes])* or *AttrsDict*) – The attributes to remove from the object. Specify an empty list for a value to delete all values.
- **current** (*LDAPObject* or *None*) – The current known state of the object. Used to ensure we don't request that the server delete attribute values that don't exist and for validation.

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

disable_validation (*disabled_validators=None*)

Returns a context manager which temporarily disables validation. If any server errors are generated, they will still be propagated.

Example:

```
from laurelin.ldap import LDAP
from laurelin.ldap.exceptions import LDAPValidationError
from laurelin.ldap.schema import SchemaValidator

with LDAP(validators=[SchemaValidator()]) as ldap:
    # make validated queries
    ldap.base.add_child('cn=foo', {<valid object>})

    try:
        ldap.base.add_child('cn=bar', {<invalid object>})
    except LDAPValidationError:
        pass

    with ldap.disable_validation(['SchemaValidator']):
        # make queries without validation
        ldap.base.add_child('cn=bar', {<invalid object>})
        # NOTE: if the object is actually invalid, a server error may still_
↪ occur

    # carry on with validation restored...
```

Parameters `disabled_validators` – Optional, a list of string class names or Validator classes to disable. By default all validators will be disabled.

Returns A context manager which temporarily disables validation

Return type *DisabledValidationContext*

static `disable_warnings()`

Prevent all LDAP warnings from being shown - default action for others

static `enable_logging(level=10)`

Enable logging output to stderr

exists (*dn*)

Simply check if a DN exists.

Parameters `dn` (*str*) – The DN to check

Returns True if the object exists, False if not

Return type `bool`

get (*dn*, *attrs=None*, ***kws*)

Get a specific object by DN.

Performs a search with *Scope.BASE* and ensures we get exactly one result.

Parameters

- `dn` (*str*) – The DN of the object to query
- `attrs` (*list[str]* or *None*) – Optional. A list of attribute names to get, defaults to all user attributes

Returns The LDAP object

Return type *LDAPObject*

Raises

- **ConnectionUnbound** – if the connection has been unbound
- **NoSearchResults** – if no results are returned
- **MultipleSearchResults** – if more than one result is returned

Additional keyword arguments are passed through into `LDAP.search()`.

get_sasl_mechs()

Query root DSE for supported SASL mechanisms.

Returns The list of server-supported mechanism names.

Return type `list[str]`

static log_warnings()

Log all LDAP warnings rather than showing them - default action for others

mod_dn (*dn*, *new_rdn*, *clean_attr=True*, *new_parent=None*, ***ctrl_kwds*)

Change the DN and possibly the location of an object in the tree. Exposes all options of the protocol-level `rfc4511.ModifyDNRequest`

Parameters

- **dn** (*str*) – The current DN of the object
- **new_rdn** (*str*) – The new RDN of the object, e.g. `cn=foo`
- **clean_attr** (*bool*) – Remove the old RDN attribute from the object when changing
- **new_parent** (*str or None*) – The DN of the new parent object, or `None` to leave the location unchanged

Returns A response object

Return type `LDAPResponse`

Raises **ConnectionUnbound** – if the connection has been unbound

Additional keyword arguments are handled as *Controls*.

modify (*dn*, *modlist*, *current=None*, ***ctrl_kwds*)

Perform a series of modify operations on an object atomically

Parameters

- **dn** (*str*) – The DN of the object to modify
- **modlist** (*list[Mod]*) – A list of *Mod* instances, e.g. `[Mod(Mod.ADD, 'someAttr', ['value1', 'value2'])]`
- **current** (`LDAPObject or None`) – The current known state of the object for use in validation

Returns A response object

Return type `LDAPResponse`

Raises

- **ConnectionUnbound** – if the connection has been unbound
- **LDAPValidationError** – if the operation fails and configured validator

Additional keyword arguments are handled as *Controls*.

move (*dn*, *new_dn*, *clean_attr=True*, ***ctrl_kwds*)

Specify a new absolute DN for an object.

Parameters

- **dn** (*str*) – The current DN of the object
- **new_dn** (*str*) – The new absolute DN of the object, e.g. cn=foo,dc=example,dc=org
- **clean_attr** (*bool*) – Remove the old RDN attribute from the object when changing

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

obj (*dn*, *attrs_dict*=None, *tag*=None, ***kws*)

Factory for LDAPObjects bound to this connection.

Note that this does not query the server. Use *LDAP.get()* to query the server for a particular DN.

Parameters

- **dn** (*str*) – The DN of the object.
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict or None*) – Optional. The object's attributes and values.
- **tag** (*str or None*) – Optional. The tag for this object. Tagged objects can be retrieved with *LDAP.tag()*.

Returns The new object bound to this connection.

Return type *LDAPObject*

Raises *TagError* – if the tag parameter is already defined

Additional keywords are passed through into the *LDAPObject* constructor.

process_ldif (*ldif_str*)

Process a basic LDIF

TODO: full RFC 2849 implementation. Missing:

- attribute options

Parameters **ldif_str** (*str*) – An RFC 2849 complying LDIF string

Returns A list with elements corresponding to the return of each described operation

Return type *list[LDAPResponse or LDAPObject]*

Raises

- *ValueError* – if the LDIF is malformed
- *LDAPError* – if an unimplemented feature is used
- *LDAPSupportError* – if a version other than 1 is specified or a critical control is undefined

recheck_sasl_mechs ()

Query the root DSE again after performing a SASL bind to check for a downgrade attack.

Raises *LDAPError* – If the downgrade attack check fails and *sasl_fatal_downgrade_check* has not been set to False.

refresh_root_dse()

Update the local copy of the root DSE, containing metadata about the directory server. The root DSE is an *LDAPObject* stored on the *root_dse* attribute.

rename(*dn*, *new_rdn*, *clean_attr=True*, ***ctrl_kwds*)

Specify a new RDN for an object without changing its location in the tree.

Parameters

- **dn** (*str*) – The current DN of the object
- **new_rdn** (*str*) – The new RDN of the object, e.g. cn=foo
- **clean_attr** (*bool*) – Remove the old RDN attribute from the object when changing

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

replace_attrs(*dn*, *attrs_dict*, *current=None*, ***ctrl_kwds*)

Replace all values on given attributes with the passed values

- Attributes not mentioned in *attrsDict* are not touched
- Attributes will be created if they do not exist
- Specifying a 0-length entry will delete all values for that attribute

Parameters

- **dn** (*str*) – The DN of the object to modify
- **attrs_dict** (*dict(str, list(str or bytes))* or *AttrsDict*) – The new attributes to set on the object
- **current** (*LDAPObject* or *None*) – The current known state of the object for use in validation

Returns A response object

Return type *LDAPResponse*

Additional keyword arguments are handled as *Controls*.

sasl_bind(*mech=None*, ***props*)

Perform a SASL bind operation.

Keywords are first taken as *Controls*. Required keyword args are dependent on the mechanism chosen.

Parameters **mech** (*str*) – The SASL mechanism name to use or *None* to negotiate best mutually supported mechanism.

Returns A response object

Return type *LDAPResponse*

Raises

- *ConnectionUnbound* – if the connection has been unbound/closed
- *ConnectionAlreadyBound* – if the connection has already been bound
- *LDAPSupportError* – if the given *mech* is not supported by the server
- *LDAPError* – if an error occurs during the bind process

search(*base_dn*, *scope*=*Scope.SUB*, *filter*=*None*, *attrs*=*None*, *search_timeout*=*None*, *limit*=0, *deref_aliases*=*None*, *attrs_only*=*False*, *fetch_result_refs*=*None*, *follow_referrals*=*None*, *filter_syntax*=*None*, ***kws*)

Sends search and return an iterator over results.

Parameters

- **base_dn** (*str*) – The DN of the base object of the search
- **scope** (*Scope*) – One of the *Scope* constants, default *Scope.SUB*. Controls the maximum depth of the search.
- **filter** (*str*) – A filter string. Objects must match the filter to be included in results. Default includes all objects and can be overridden globally by defining *LDAP.DEFAULT_FILTER*.
- **attrs** (*list[str]*) – A list of attribute names to include for each object. Default includes all user attributes. Use *['*', '+']* to get all user and all operational attributes.
- **search_timeout** (*int*) – The number of seconds the server should spend performing the search. Partial results will be returned if the server times out. The default can be set per connection by passing the *search_timeout* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SEARCH_TIMEOUT*.
- **limit** (*int*) – The maximum number of objects to return.
- **deref_aliases** (*DerefAliases*) – One of the *DerefAliases* constants. This instructs the server what to do when it encounters an alias object. The default can be set per connection by passing the *deref_aliases* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_DEREF_ALIASES*.
- **attrs_only** (*bool*) – Default *False*. Set to *True* to only obtain attribute names and not any attribute values.
- **fetch_result_refs** (*bool*) – When the server returns a result which is a reference to an object on another server, automatically attempt to fetch the remote object and include it in the iterated results. The default can be set per connection by passing the *fetch_result_refs* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_FETCH_RESULT_REFS*.
- **follow_referrals** (*bool*) – When the server knows that the base object is present on another server, follow the referral and perform the search on the other server. The default can be set per connection by passing the *follow_referrals* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_FOLLOW_REFERRALS*.
- **filter_syntax** (*FilterSyntax*) – Select which filter syntax to use to parse the filter. The default can be set per connection by passing the *default_filter_syntax* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_FILTER_SYNTAX*.

Returns An iterator over the results of the search. May yield *LDAPObject* or possibly *SearchReferenceHandle* if *fetch_result_refs* is *False*.

Additional keywords are handled as *Controls* first and then passed through into *LDAP.obj()*.

This method may also be used as a context manager. If all results have not been read, the operation will automatically be abandoned when the context manager exits. You can also raise *Abandon* to abandon all results immediately and cleanly exit the context manager. You can also call *SearchResultHandle.abandon()* to abandon results.

Example:


```
# Dump the whole tree
with LDAP() as ldap:
    with ldap.base.search() as search:
        for result in search:
            print(result.format_ldif())
```

send_extended_request (*oid*, *value=None*, ***kws*)

Send an extended request, returns instance of `ExtendedResponseHandle`

This is mainly meant to be called by other built-in methods and client extensions. Requires handling of raw pyasn1 protocol objects.

Parameters

- **oid** (*str*) – The OID of the extension. Must be declared as supported by the server in the root DSE.
- **value** (*str* or *bytes* or *None*) – The request value (optional)

Returns An iterator yielding tuples of the form (`rfc4511.IntermediateResponse`, `rfc4511.Controls`) or (`rfc4511.ExtendedResponse`, `rfc4511.Controls`).

Return type *ExtendedResponseHandle*

Raises

- **LDAPSupportError** – if the OID is not listed in the `supportedExtension` attribute of the root DSE
- **TypeError** – if the *value* parameter is not a valid type

Additional keyword arguments are handled as *Controls* and then passed through into the `ExtendedResponseHandle` constructor.

simple_bind (*username=""*, *password=""*, ***ctrl_kws*)

Performs a simple bind operation

Leave arguments as their default (empty strings) to attempt an anonymous simple bind

Additional keywords are used as *Controls*.

Parameters

- **username** (*str*) – Bind DN/username or empty string for anonymous
- **password** (*str*) – Password to bind with or empty string for anonymous

Returns A response object

Return type *LDAPResponse*

Raises

- **ConnectionUnbound** – if the connection has been unbound/closed
- **ConnectionAlreadyBound** – if the connection has already been bound

start_tls (*verify=None*, *ca_file=None*, *ca_path=None*, *ca_data=None*)

Perform the StartTLS extended operation. This will instruct the server to begin encrypting this socket connection with TLS/SSL.

Parameters

- **verify** (*bool*) – Set to False to disable verification of the remote certificate. You can set the default per-connection by passing the *ssl_verify* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_VERIFY*.

- **ca_file** (*str*) – Path to PEM-formatted concatenated CA certificates file. You can set the default per-connection by passing the *ssl_ca_file* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_CA_FILE*.
- **ca_path** (*str*) – Path to directory with CA certs under hashed file names. See https://www.openssl.org/docs/man1.1.0/ssl/SSL_CTX_load_verify_locations.html for more information about the format of this directory. You can set the default per-connection by passing the *ssl_ca_path* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_CA_PATH*.
- **ca_data** (*str* or *bytes*) – An ASCII string of one or more PEM-encoded certs or a bytes object containing DER-encoded certificates. You can set the default per-connection by passing the *ssl_ca_data* keyword to the *LDAP* constructor, or set the global default by defining *LDAP.DEFAULT_SSL_CA_DATA*.

Return type *None*

tag (*tag*)

Get a tagged object.

Parameters **tag** (*str*) – The tag name to retrieve

Returns The object created with the given tag

Return type *LDAPObject*

Raises *TagError* – if the given tag is not defined

unbind (*force=False*)

Send an unbind request and close the socket.

Parameters **force** (*bool*) – Unbind and close the socket even if other objects still hold a reference to it.

Raises *ConnectionUnbound* – if the connection has already been unbound

validate_modify (*dn, modlist, current=None*)

Run all configured validators for the given modify operation

Parameters

- **dn** (*str*) – The DN of the object being modified
- **modlist** (*list [Mod]*) – The sequence of changes to be performed
- **current** (*LDAPObject*) – The current known state of the object

Return type *None*

Raises *LDAPValidationError* – if any validator fails the operation

validate_object (*obj, write=True*)

Run all configured validators for the given object.

Parameters

- **obj** (*LDAPObject*) – The object to validate
- **write** (*bool*) – True if this is for a write operation (e.g. an add)

Return type *None*

Raises *LDAPValidationError* – if any validator fails the object

who_am_i (***ctrl_kwds*)

Perform the “Who Am I?” extended operation. This will confirm the identity that the connection is bound to.

Returns A string describing the bound identity. One common form is “dn:cn=foo,dc=example,dc=org” but this will vary by server configuration and bind type/parameters.

Return type *str*

Additional keyword arguments are handled as *Controls*.

class `laurelin.ldap.LDAPURI` (*uri*)

Bases: *object*

Represents a parsed LDAP URI as specified in RFC4516

Supported extensions:

- “StartTLS”

Variables

- **scheme** (*str*) – urlparse standard
- **netloc** (*str*) – urlparse standard
- **host_uri** (*str*) – scheme://netloc for use with LDAPSocket
- **dn** (*str*) – Distinguished name
- **attrs** (*list[str]*) – list
- **scope** (*Scope*) – one of the *Scope* constants
- **filter** (*str*) – The filter string
- **starttls** (*bool*) – True if StartTLS was requested

DEFAULT_ATTRS = ['*']

DEFAULT_FILTER = '(objectClass=*)'

DEFAULT_SCOPE = *Scope.BASE*

DEFAULT_STARTTLS = *False*

search (***kwds*)

Perform the search operation described by the parsed URI

First opens a new connection with connection reuse disabled, then performs the search, and unbinds the connection. Server must allow anonymous read.

Additional keyword arguments are passed through into *LDAP.search()*.

class `laurelin.ldap.Scope`

Bases: *object*

Scope constants. These instruct the server how far to take a search, relative to the base object

BASE = *Scope.BASE*

Only search the base object

ONE = *Scope.ONE*

Search the base object and its immediate children

ONELEVEL = *Scope.ONE*

SUB = Scope.SUB

Search the base object and all of its descendants

SUBTREE = Scope.SUB

static constant (c)

translate constants to RFC4516 URL scope string

static string (str)

translate RFC4516 URL scope strings to constant

class laurelin.ldap.DerefAliases

Bases: *object*

DerefAliases constants. These instruct the server when to automatically resolve an alias object, rather than return the alias object itself

ALWAYS = DerefAliases.ALWAYS

dereferences both the search base object and results

BASE = DerefAliases.BASE

dereferences the search base object, but not search results

NEVER = DerefAliases.NEVER

always return the alias object

SEARCH = DerefAliases.SEARCH

dereferences search results, but not the base object itself

class laurelin.ldap.FilterSyntax

Bases: *object*

Filter syntax selection constants. Used to determine which filter syntax to use when parsing a search filter.

SIMPLE = FilterSyntax.SIMPLE

STANDARD = FilterSyntax.STANDARD

UNIFIED = FilterSyntax.UNIFIED

static string (str)

Convert filter syntax string to constant

class laurelin.ldap.critical (value)

Bases: *object*

used to mark controls with criticality

class laurelin.ldap.optional (value)

Bases: *object*

used to mark controls as not having criticality

exception laurelin.ldap.LDAPError

Bases: *Exception*

Base class for all exceptions raised by laurelin

exception laurelin.ldap.NoSearchResults

Bases: *laurelin.ldap.exceptions.UnexpectedSearchResults*

Got no search results when one or more was required

exception laurelin.ldap.Abandon

Bases: *Exception*

Can be raised to cleanly exit a context manager and abandon unread results

```
class laurelin.ldap.LDAPObject (dn,          attrs_dict=None,          ldap_conn=None,          rela-
                               tive_search_scope=Scope.SUB, rdn_attr=None)
Bases: laurelin.ldap.attrsdict.AttrsDict, laurelin.ldap.extensible.Extensible
```

Represents a single object with optional server affinity.

Many methods will raise an exception if used without a server connection. To instantiate an *LDAPObject* bound to a server connection, use *LDAP.obj()*.

Attributes and values are stored using the mapping interface inherited from *AttrsDict*, where dict keys are case-insensitive attribute names, and dict values are a list of attribute values.

Value lists are automatically wrapped in *AttrValueList*. This allows the use of any schema-defined matching and syntax rules for the attribute type in list operations.

Parameters

- **dn** (*str*) – The DN of the object
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict or None*) – The object's attributes
- **ldap_conn** (*LDAP or None*) – The optional LDAP connection to use
- **relative_search_scope** (*Scope*) – One of the *Scope* constants, this is the default scope used when using this object's *LDAPObject.search()* method. New objects created below this one will inherit this attribute by default. This attribute also defines the behavior of *LDAPObject.find()*.
- **rdn_attr** (*str or None*) – The default attribute name used in RDN's for descendants of this object. If specified, this allows you to only specify the value for methods that have an *rdn* argument. You can always specify a full *attr=value* for *rdn* arguments as well to override this behavior. New objects created below this one will inherit this attribute by default.

```
add_attrs (attrs_dict, **ctrl_kwds)
```

Add new attribute values to this object.

Parameters **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict*) – The new attributes to add to the object

Return type *None*

Additional keywords are passed through into *LDAPObject.modify()*.

```
add_child (rdn, attrs_dict, **kwds)
```

Create a new object below this one.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object
- **attrs_dict** (*dict(str, list[str or bytes]) or AttrsDict or None*) – The attributes for the object

Returns The new object

Return type *LDAPObject*

Additional keyword arguments are passed through into *LDAP.add()*

```
compare (attr, value)
```

Ask the server if this object has a matching attribute value. The comparison will take place following the schema-defined matching rules and syntax rules.

Parameters

- **attr** (*str*) – The attribute name
- **value** (*str*) – The assertion value

Returns A response object, `bool()` evaluating to the result of the comparison

Return type *CompareResponse*

Raises *RuntimeError* – if this object is not bound to an LDAP connection

delete (***ctrl_kwds*)

Delete the entire object from the server, and render this instance useless.

Additional keywords are passed through into *LDAP.delete()*.

Return type *None*

Raises *RuntimeError* – if this object is not bound to an LDAP connection

delete_attrs (*attrs_dict, **ctrl_kwds*)

Delete specific attribute values given in *attrs_dict*. Specifying a zero-length list for any attribute will delete all values for that attribute.

Parameters **attrs_dict** (*dict(str, list[str or bytes])* or *AttrsDict*) –
The attributes to delete from the object

Return type *None*

Additional keywords are passed through into *LDAPObject.modify()*.

delete_child (*rdn, **ctrl_kwds*)

Delete a child object below this one.

Parameters **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object

Returns The *LDAPResponse* from the delete operation

Return type *LDAPResponse*

Additional keyword arguments are treated as controls.

find (*rdn, attrs=None, **kwds*)

Obtain a single object below this one with the most efficient means possible.

The strategy used is based on the *relative_search_scope* property of this object.

- If it is *Scope.BASE*, this method will always raise an *LDAPError*.
- If it is *Scope.ONE*, then the absolute DN for the child object will be constructed, and a *Scope.BASE* search will be performed to get the object.
- If it is *Scope.SUB*, then a subtree search will be performed below this object, using the RDN as a search filter.

Additional keywords are passed through into *LDAPObject.search()*.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object
- **attrs** (*list[str]*) – Optional. The list of attribute names to obtain.

Returns The LDAP object

Return type *LDAPObject*

Raises

- **LDAPError** – if this object’s `relative_search_scope` is `Scope.BASE`.
- **NoSearchResults** – if no object could be found matching `rdn`.
- **MultipleSearchResults** – if more than one object was found.
- **RuntimeError** – if this object is not bound to an LDAP connection
- **ValueError** – if the `relative_search_scope` is set to an invalid value.

format_ldif()

Format the object as an LDIF string.

Returns The object encoded as an LDIF.

Return type `str`

get_child(*rdn*, *attrs=None*, *kws*)**

Query the server for a child object.

Parameters

- **rdn** (`str`) – The RDN, or RDN value if `rdn_attr` is defined for this object
- **attrs** (`list[str]` or `None`) – The list of attributes to query

Returns The object populated with data from the server

Return type `LDAPObject`

Raises **RuntimeError** – if this object is not bound to an LDAP connection

Additional keywords are passed through into `LDAP.search()` and `LDAPObject`

has_object_class(*object_class*)

A convenience method which checks if this object has a particular objectClass. May query the server for the objectClass attribute if it is not yet known.

Parameters **object_class** – The objectClass to check for.

Returns True if the objectClass is present, False otherwise

Return type `bool`

mod_dn(*new_rdn*, *clean_attr=True*, *new_parent=None*, *ctrl_kws*)**

Change the object DN, and possibly its location in the tree.

Parameters

- **new_rdn** (`str`) – The new RDN of the object
- **clean_attr** (`bool`) – Optional, default True. Remove the attribute associated with the RDN when changing it.
- **new_parent** (`str`) – Optional. The absolute DN of the object’s new parent.

Return type `None`

Raises **RuntimeError** – if this object is not bound to an LDAP connection

Additional keywords are passed through into `LDAP.mod_dn()`.

mod_transaction()

Begin a modify transaction on this object. Important: This IS NOT an RFC 5805 transaction.

Return type `ModTransactionObject`

modify(*modlist*, *ctrl_kws*)**

Perform a series of modify operations on this object atomically.

Parameters `modlist` (*list* [*Mod*]) – A list of *Mod* instances, e.g. [`Mod(Mod.ADD, 'someAttr', ['value1', 'value2'])`]

Return type `None`

Raises `RuntimeError` – if this object is not bound to an LDAP connection

Additional keywords are passed through into `LDAP.modify()`.

move (*new_dn*, *clean_attr=True*, ***ctrl_kwds*)

Specify the complete new absolute DN for this object.

Parameters

- **new_dn** (*str*) – The new absolute DN for the object
- **clean_attr** (*bool*) – Optional, default True. Remove the attribute associated with the RDN when changing it.

Return type `None`

Additional keywords are passed through into `LDAPObject.mod_dn()`.

obj (*rdn*, *attrs_dict=None*, *tag=None*, ***kwds*)

Create a new object below this one.

Parameters

- **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object
- **attrs_dict** (*dict*(*str*, *list*[*str* or *bytes*]) or *AttrsDict* or *None*) – The attributes for the object
- **tag** (*str* or *None*) – Optional tag for the object

Returns The new object

Return type `LDAPObject`

Raises `LDAPError` – if a *tag* is specified but this object is not bound to an LDAP connection

Additional keywords are passed through into `LDAP.obj()` or the `LDAPObject` constructor.

rdn (*rdn*)

Return an absolute DN from an RDN or RDN value

Parameters **rdn** (*str*) – The RDN, or RDN value if *rdn_attr* is defined for this object

Returns The absolute DN

Return type `str`

refresh (*attrs=None*)

Query the server to update the attributes on this object.

Parameters **attrs** (*list*[*str*]) – Optional. A list of attribute names to query. If not specified, will query the server for all user attributes.

Return type `None`

Raises `RuntimeError` – if this object is not bound to an LDAP connection

refresh_all ()

Query the server to update all user and operational attributes on this object.

Return type `None`

Raises `RuntimeError` – if this object is not bound to an LDAP connection

refresh_missing (*attrs*)

Potentially query the server for any listed attributes that are not yet defined on this object. If no listed attributes aren't defined, the query will not be performed. If a subset of the list is undefined, only those attributes will be queried.

Parameters *attrs* (*list[str]*) – A list of attribute names to check, and possibly query for.

Return type *None*

rename (*new_rdn*, *clean_attr=True*, ***ctrl_kws*)

Change the object's RDN without changing it's location in the tree.

Parameters

- **new_rdn** (*str*) – The new RDN of the object
- **clean_attr** (*bool*) – Optional, default True. Remove the attribute associated with the RDN when changing it.

Return type *None*

Additional keywords are passed through into *LDAPObject.mod_dn()*.

replace_attrs (*attrs_dict*, ***ctrl_kws*)

Replace all values on the given attributes with the passed values.

Parameters *attrs_dict* (*dict(str, list[str or bytes])* or *AttrsDict*) – The new attributes to set on the object

Return type *None*

Additional keywords are passed through into *LDAPObject.modify()*.

search (*filter=None*, *attrs=None*, ***kws*)

Perform a search below this object.

Parameters

- **filter** (*str*) – Optional. The filter string to use to filter returned objects.
- **attrs** (*list[str]*) – Optional. The list of attribute names to retrieve.

Returns An iterator over *LDAPObject* and possibly *SearchReferenceHandle*. See *LDAP.search()* for more details.

Return type *SearchResultHandle*

Additional keywords are passed through into *LDAP.search()*.

validate ()

Validate the object, assuming all attributes are present locally

validate_modify (*modlist*)

Validate a modification list.

Parameters *modlist* (*list[Mod]*) – The list of modify operations to validate.

laurelin.ldap.escape (*text*)

Escape special characters

class *laurelin.ldap.Mod* (*op*, *attr*, *vals*)

Bases: *object*

Describes a single modify operation

ADD = **Mod.ADD**

DELETE = **Mod.DELETE**

REPLACE = **Mod.REPLACE**

static op_to_string(*op*)

Convert one of the *Mod* constants to a string, e.g. “ADD”, “REPLACE”, “DELETE”.

static string(*op*)

Translte LDIF changetype strings to constant. e.g. “replace” -> *Mod.REPLACE*

laurelin.ldap.dc(*domain*)

Convert a DNS dotted domain name to a DN with domain components

laurelin.ldap.domain(*dc*)

Convert a DN with domain components to a DNS dotted domain name

Warning: Testing has been moved to docker using public images. Check `.travis.yml` for details. This page is maintained for historical documentation purposes.

9.1 System

- Digital Ocean VPS with Debian 7.9
- OpenLDAP 2.4.31
- Cyrus SASL 2.1.25
- 389 Directory Server 1.3.6

9.2 SASL

9.2.1 SASL config Idif

```
dn: cn=config
changetype: modify
replace: olcAuthzRegexp
olcAuthzRegexp: uid=([^\,]+),.* cn=$1,dc=example,dc=org
-
add: olcSaslAuxprops
olcSaslAuxprops: sasldb
-
add: olcSaslRealm
olcSaslRealm: example.org
-
add: olcSaslHost
```

(continues on next page)

(continued from previous page)

```
olcSaslHost: example.org
-
```

9.2.2 Adding sasl user password with

```
saslpasswd2 -u example.org -c $USER
```

9.2.3 SASL auth control test case

```
% ldapwhoami -Y DIGEST-MD5 -U admin -H ldap://127.0.0.1
SASL/DIGEST-MD5 authentication started
Please enter your password:
SASL username: admin
SASL SSF: 128
SASL data security layer installed.
dn:cn=admin,dc=example,dc=org
```

9.3 LDAPS/StartTLS

- Certs set up following this [Stack Overflow answer](#).
- Configured OpenLDAP as follows:

```
dn: cn=config
changetype: modify
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /certs/serverkey.pem
-
replace: olcTLSCertificateFile
olcTLSCertificateFile: /certs/servercert.pem
-
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /certs/cacert.pem
```

- Added `ldaps://127.0.0.1:636` to `SLAPD_SERVICES` in `/etc/default/slapd`

CHAPTER 10

Indices and tables

- `genindex`
- `modindex`
- `search`

I

- `laurelin.extensions.descattrs`, 17
- `laurelin.extensions.pagedresults`, 19
- `laurelin.ldap`, 80
 - `laurelin.ldap.attrbutetype`, 35
 - `laurelin.ldap.attrsdict`, 38
 - `laurelin.ldap.attrvaluelist`, 38
 - `laurelin.ldap.base`, 39
 - `laurelin.ldap.config`, 53
 - `laurelin.ldap.constants`, 55
 - `laurelin.ldap.exceptions`, 56
 - `laurelin.ldap.extensible`, 58
 - `laurelin.ldap.filter`, 58
 - `laurelin.ldap.ldapobject`, 59
 - `laurelin.ldap.modify`, 66
 - `laurelin.ldap.net`, 66
 - `laurelin.ldap.objectclass`, 68
 - `laurelin.ldap.rules`, 69
 - `laurelin.ldap.schema`, 71
 - `laurelin.ldap.validation`, 80

A

Abandon, 56, 94
 abandon() (laurelin.Ldap.base.ResponseHandle method), 53
 activate_extension() (laurelin.Ldap.base.LDAP static method), 42
 activate_extension() (laurelin.Ldap.LDAP static method), 83
 activate_extensions() (in module laurelin.Ldap.config), 53
 ADD (laurelin.Ldap.Mod attribute), 99
 ADD (laurelin.Ldap.modify.Mod attribute), 66
 add() (laurelin.Ldap.base.LDAP method), 42
 add() (laurelin.Ldap.LDAP method), 83
 add_attrs() (laurelin.Ldap.base.LDAP method), 42
 add_attrs() (laurelin.Ldap.LDAP method), 83
 add_attrs() (laurelin.Ldap.LDAPObject method), 95
 add_attrs() (laurelin.Ldap.Ldapobject.LDAPObject method), 59
 add_child() (laurelin.Ldap.LDAPObject method), 95
 add_child() (laurelin.Ldap.Ldapobject.LDAPObject method), 59
 add_child() (laurelin.Ldap.Ldapobject.ModTransactionObject method), 65
 add_if_not_exists() (laurelin.Ldap.base.LDAP method), 43
 add_if_not_exists() (laurelin.Ldap.LDAP method), 83
 add_or_mod_add_if_exists() (laurelin.Ldap.base.LDAP method), 43
 add_or_mod_add_if_exists() (laurelin.Ldap.LDAP method), 84
 add_or_mod_replace_if_exists() (laurelin.Ldap.base.LDAP method), 43
 add_or_mod_replace_if_exists() (laurelin.Ldap.LDAP method), 84
 AddModlist() (in module laurelin.Ldap.modify), 66
 allowed_attr() (laurelin.Ldap.objectclass.ExtensibleObjectClass method), 68
 allowed_attr() (laurelin.Ldap.objectclass.ObjectClass method), 69

ALWAYS (laurelin.Ldap.constants.DerefAliases attribute), 55
 ALWAYS (laurelin.Ldap.DerefAliases attribute), 94
 AttributeType (class in laurelin.Ldap.attributetype), 35
 AttributeTypeDescription (class in laurelin.Ldap.schema), 71
 AttrsDict (class in laurelin.Ldap.attrsdict), 38
 AttrValueList (class in laurelin.Ldap.attrvaluelist), 38

B

BASE (laurelin.Ldap.constants.DerefAliases attribute), 56
 BASE (laurelin.Ldap.constants.Scope attribute), 56
 BASE (laurelin.Ldap.DerefAliases attribute), 94
 BASE (laurelin.Ldap.Scope attribute), 93
 Binary (class in laurelin.Ldap.schema), 71
 BitString (class in laurelin.Ldap.schema), 71
 bitStringMatch (class in laurelin.Ldap.schema), 77
 Boolean (class in laurelin.Ldap.schema), 71
 booleanMatch (class in laurelin.Ldap.schema), 77

C

caseExactIA5Match (class in laurelin.Ldap.schema), 77
 caseExactMatch (class in laurelin.Ldap.schema), 77
 caseIgnoreIA5Match (class in laurelin.Ldap.schema), 77
 caseIgnoreListMatch (class in laurelin.Ldap.schema), 78
 caseIgnoreMatch (class in laurelin.Ldap.schema), 78
 Certificate (class in laurelin.Ldap.schema), 72
 check_hostname() (laurelin.Ldap.net.LDAPSocket method), 67
 close() (laurelin.Ldap.base.LDAP method), 44
 close() (laurelin.Ldap.LDAP method), 84
 close() (laurelin.Ldap.net.LDAPSocket method), 67
 commit() (laurelin.Ldap.Ldapobject.ModTransactionObject method), 65
 compare() (laurelin.Ldap.base.LDAP method), 44
 compare() (laurelin.Ldap.LDAP method), 85
 compare() (laurelin.Ldap.LDAPObject method), 95
 compare() (laurelin.Ldap.Ldapobject.LDAPObject method), 60

CompareResponse (class in laurelin.ldap.base), 39
ConnectionAlreadyBound, 56
ConnectionUnbound, 56
constant() (laurelin.ldap.constants.Scope static method), 56
constant() (laurelin.ldap.Scope static method), 94
Control (class in laurelin.ldap.controls), 28
count() (laurelin.ldap.attrvaluelist.AttrValueList method), 39
CountryString (class in laurelin.ldap.schema), 72
create_connection() (in module laurelin.ldap.config), 53
critical (class in laurelin.ldap), 28, 94

D

dc() (in module laurelin.ldap), 100
deepcopy() (laurelin.ldap.attrsdict.AttrsDict method), 38
DEFAULT_ATTRS (laurelin.ldap.base.LDAPURI attribute), 52
DEFAULT_ATTRS (laurelin.ldap.LDAPURI attribute), 93
DEFAULT_BASE_DN (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_BASE_DN (laurelin.ldap.LDAP attribute), 82
DEFAULT_CONNECT_TIMEOUT (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_CONNECT_TIMEOUT (laurelin.ldap.LDAP attribute), 82
DEFAULT_CRITICALITY (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_CRITICALITY (laurelin.ldap.LDAP attribute), 82
DEFAULT_DEREF_ALIASES (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_DEREF_ALIASES (laurelin.ldap.LDAP attribute), 82
DEFAULT_ERROR_EMPTY_LIST (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_ERROR_EMPTY_LIST (laurelin.ldap.LDAP attribute), 82
DEFAULT_FETCH_RESULT_REFS (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_FETCH_RESULT_REFS (laurelin.ldap.LDAP attribute), 82
DEFAULT_FILTER (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_FILTER (laurelin.ldap.base.LDAPURI attribute), 52
DEFAULT_FILTER (laurelin.ldap.LDAP attribute), 82
DEFAULT_FILTER (laurelin.ldap.LDAPURI attribute), 93
DEFAULT_FILTER_SYNTAX (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_FILTER_SYNTAX (laurelin.ldap.LDAP attribute), 82

DEFAULT_FOLLOW_REFERRALS (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_FOLLOW_REFERRALS (laurelin.ldap.LDAP attribute), 82
DEFAULT_IGNORE_EMPTY_LIST (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_IGNORE_EMPTY_LIST (laurelin.ldap.LDAP attribute), 82
DEFAULT_REUSE_CONNECTION (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_REUSE_CONNECTION (laurelin.ldap.LDAP attribute), 82
DEFAULT_SASL_FATAL_DOWNGRADE_CHECK (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_SASL_FATAL_DOWNGRADE_CHECK (laurelin.ldap.LDAP attribute), 82
DEFAULT_SASL_MECH (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_SASL_MECH (laurelin.ldap.LDAP attribute), 82
DEFAULT_SCOPE (laurelin.ldap.base.LDAPURI attribute), 52
DEFAULT_SCOPE (laurelin.ldap.LDAPURI attribute), 93
DEFAULT_SEARCH_TIMEOUT (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_SEARCH_TIMEOUT (laurelin.ldap.LDAP attribute), 82
DEFAULT_SERVER (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_SERVER (laurelin.ldap.LDAP attribute), 82
DEFAULT_SSL_CA_DATA (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_SSL_CA_DATA (laurelin.ldap.LDAP attribute), 82
DEFAULT_SSL_CA_FILE (laurelin.ldap.base.LDAP attribute), 41
DEFAULT_SSL_CA_FILE (laurelin.ldap.LDAP attribute), 82
DEFAULT_SSL_CA_PATH (laurelin.ldap.base.LDAP attribute), 42
DEFAULT_SSL_CA_PATH (laurelin.ldap.LDAP attribute), 82
DEFAULT_SSL_VERIFY (laurelin.ldap.base.LDAP attribute), 42
DEFAULT_SSL_VERIFY (laurelin.ldap.LDAP attribute), 82
DEFAULT_STARTTLS (laurelin.ldap.base.LDAPURI attribute), 52
DEFAULT_STARTTLS (laurelin.ldap.LDAPURI attribute), 93
DEFAULT_STRICT_MODIFY (laurelin.ldap.base.LDAP attribute), 42
DEFAULT_STRICT_MODIFY (laurelin.ldap.LDAP attribute), 82

- tribute), 82
- DEFAULT_VALIDATORS (laurelin.ldap.base.LDAP attribute), 42
- DEFAULT_VALIDATORS (laurelin.ldap.LDAP attribute), 82
- DEFAULT_WARN_EMPTY_LIST (laurelin.ldap.base.LDAP attribute), 42
- DEFAULT_WARN_EMPTY_LIST (laurelin.ldap.LDAP attribute), 82
- default_warnings() (laurelin.ldap.base.LDAP static method), 44
- default_warnings() (laurelin.ldap.LDAP static method), 85
- DefaultAttributeType (class in laurelin.ldap.attributetype), 36
- DefaultMatchingRule (class in laurelin.ldap.attributetype), 37
- DefaultObjectClass (class in laurelin.ldap.objectclass), 68
- DefaultSyntaxRule (class in laurelin.ldap.attributetype), 37
- DELETE (laurelin.ldap.Mod attribute), 99
- DELETE (laurelin.ldap.modify.Mod attribute), 66
- delete() (laurelin.ldap.base.LDAP method), 44
- delete() (laurelin.ldap.LDAP method), 85
- delete() (laurelin.ldap.LDAPObject method), 96
- delete() (laurelin.ldap.Ldapobject.LDAPObject method), 60
- delete() (laurelin.ldap.Ldapobject.ModTransactionObject method), 65
- DELETE_ALL (laurelin.ldap.base.LDAP attribute), 42
- DELETE_ALL (laurelin.ldap.LDAP attribute), 82
- delete_attrs() (laurelin.ldap.base.LDAP method), 44
- delete_attrs() (laurelin.ldap.LDAP method), 85
- delete_attrs() (laurelin.ldap.LDAPObject method), 96
- delete_attrs() (laurelin.ldap.Ldapobject.LDAPObject method), 60
- delete_child() (laurelin.ldap.LDAPObject method), 96
- delete_child() (laurelin.ldap.Ldapobject.LDAPObject method), 60
- delete_child() (laurelin.ldap.Ldapobject.ModTransactionObject method), 65
- DeleteModlist() (in module laurelin.ldap.modify), 66
- DeliveryMethod (class in laurelin.ldap.schema), 72
- DerefAliases (class in laurelin.ldap), 94
- DerefAliases (class in laurelin.ldap.constants), 55
- DESC (laurelin.ldap.rules.SyntaxRule attribute), 70
- DESC (laurelin.ldap.schema.AttributeTypeDescription attribute), 71
- DESC (laurelin.ldap.schema.Binary attribute), 71
- DESC (laurelin.ldap.schema.BitString attribute), 71
- DESC (laurelin.ldap.schema.Boolean attribute), 71
- DESC (laurelin.ldap.schema.Certificate attribute), 72
- DESC (laurelin.ldap.schema.CountryString attribute), 72
- DESC (laurelin.ldap.schema.DeliveryMethod attribute), 72
- DESC (laurelin.ldap.schema.DirectoryString attribute), 72
- DESC (laurelin.ldap.schema.DistinguishedName attribute), 73
- DESC (laurelin.ldap.schema.DITContentRuleDescription attribute), 72
- DESC (laurelin.ldap.schema.DITStructureRuleDescription attribute), 72
- DESC (laurelin.ldap.schema.EnhancedGuide attribute), 73
- DESC (laurelin.ldap.schema.FacsimileTelephoneNumber attribute), 73
- DESC (laurelin.ldap.schema.Fax attribute), 73
- DESC (laurelin.ldap.schema.GeneralizedTime attribute), 73
- DESC (laurelin.ldap.schema.Guide attribute), 74
- DESC (laurelin.ldap.schema.IA5String attribute), 74
- DESC (laurelin.ldap.schema.Integer attribute), 74
- DESC (laurelin.ldap.schema.JPEG attribute), 74
- DESC (laurelin.ldap.schema.LDAPSyntaxDescription attribute), 74
- DESC (laurelin.ldap.schema.MatchingRuleDescription attribute), 75
- DESC (laurelin.ldap.schema.MatchingRuleUseDescription attribute), 75
- DESC (laurelin.ldap.schema.NameAndOptionalUID attribute), 75
- DESC (laurelin.ldap.schema.NameFormDescription attribute), 75
- DESC (laurelin.ldap.schema.NumericString attribute), 75
- DESC (laurelin.ldap.schema.ObjectClassDescription attribute), 75
- DESC (laurelin.ldap.schema.OctetString attribute), 75
- DESC (laurelin.ldap.schema.OID attribute), 75
- DESC (laurelin.ldap.schema.OtherMailbox attribute), 76
- DESC (laurelin.ldap.schema.PostalAddress attribute), 76
- DESC (laurelin.ldap.schema.PrintableString attribute), 76
- DESC (laurelin.ldap.schema.SubstringAssertion attribute), 76
- DESC (laurelin.ldap.schema.TelephoneNumber attribute), 76
- DESC (laurelin.ldap.schema.TeletextTerminalIdentifier attribute), 77
- DESC (laurelin.ldap.schema.TelexNumber attribute), 77
- DirectoryString (class in laurelin.ldap.schema), 72
- directoryStringFirstComponentMatch (class in laurelin.ldap.schema), 78
- disable_validation() (laurelin.ldap.base.LDAP method), 45
- disable_validation() (laurelin.ldap.LDAP method), 85
- disable_warnings() (laurelin.ldap.base.LDAP static method), 45
- disable_warnings() (laurelin.ldap.LDAP static method), 85

- 86
DisabledValidationContext (class in laurelin.Ldap.validation), 80
DistinguishedName (class in laurelin.Ldap.schema), 73
distinguishedNameMatch (class in laurelin.Ldap.schema), 78
DITContentRuleDescription (class in laurelin.Ldap.schema), 72
DITStructureRuleDescription (class in laurelin.Ldap.schema), 72
do_match() (laurelin.Ldap.attributeType.DefaultMatchingRule method), 37
do_match() (laurelin.Ldap.rules.EqualityMatchingRule method), 69
do_match() (laurelin.Ldap.rules.MatchingRule method), 70
do_match() (laurelin.Ldap.schema.distinguishedNameMatch method), 78
do_match() (laurelin.Ldap.schema.generalizedTimeMatch method), 78
domain() (in module laurelin.Ldap), 100
- ## E
- enable_logging() (laurelin.Ldap.base.LDAP static method), 45
enable_logging() (laurelin.Ldap.LDAP static method), 86
EnhancedGuide (class in laurelin.Ldap.schema), 73
equality (laurelin.Ldap.attributeType.AttributeType attribute), 36
equality (laurelin.Ldap.attributeType.DefaultAttributeType attribute), 36
EqualityMatchingRule (class in laurelin.Ldap.rules), 69
escape() (in module laurelin.Ldap), 99
escape() (in module laurelin.Ldap.filter), 58
exists() (laurelin.Ldap.base.LDAP method), 45
exists() (laurelin.Ldap.LDAP method), 86
EXTEND() (laurelin.Ldap.extensible.Extensible class method), 58
ExtendedResponseHandle (class in laurelin.Ldap.base), 39
Extensible (class in laurelin.Ldap.extensible), 58
ExtensibleObjectClass (class in laurelin.Ldap.objectclass), 68
- ## F
- FacsimileTelephoneNumber (class in laurelin.Ldap.schema), 73
Fax (class in laurelin.Ldap.schema), 73
fetch() (laurelin.Ldap.base.SearchReferenceHandle method), 53
FilterSyntax (class in laurelin.Ldap), 94
FilterSyntax (class in laurelin.Ldap.constants), 56
find() (laurelin.Ldap.LDAPObject method), 96
find() (laurelin.Ldap.Ldapobject.LDAPObject method), 60
format_ldif() (laurelin.Ldap.LDAPObject method), 97
format_ldif() (laurelin.Ldap.Ldapobject.LDAPObject method), 61
format_mod_ldif() (laurelin.Ldap.Ldapobject.ModTransactionObject method), 65
- ## G
- GeneralizedTime (class in laurelin.Ldap.schema), 73
generalizedTimeMatch (class in laurelin.Ldap.schema), 78
get() (laurelin.Ldap.base.LDAP method), 45
get() (laurelin.Ldap.LDAP method), 86
get_attr() (laurelin.Ldap.attrsdict.AttrsDict method), 38
get_attribute_type() (in module laurelin.Ldap.attributeType), 37
get_child() (laurelin.Ldap.LDAPObject method), 97
get_child() (laurelin.Ldap.Ldapobject.LDAPObject method), 61
get_matching_rule() (in module laurelin.Ldap.rules), 71
get_object_class() (in module laurelin.Ldap.objectclass), 69
get_sasl_mechs() (laurelin.Ldap.base.LDAP method), 46
get_sasl_mechs() (laurelin.Ldap.LDAP method), 87
get_syntax_rule() (in module laurelin.Ldap.rules), 71
Guide (class in laurelin.Ldap.schema), 74
- ## H
- handle() (laurelin.Ldap.controls.Control method), 29
has_object_class() (laurelin.Ldap.LDAPObject method), 97
has_object_class() (laurelin.Ldap.Ldapobject.LDAPObject method), 61
- ## I
- IA5String (class in laurelin.Ldap.schema), 74
index() (laurelin.Ldap.attributeType.AttributeType method), 36
index() (laurelin.Ldap.attributeType.DefaultAttributeType method), 36
index() (laurelin.Ldap.attrvaluelist.AttrValueList method), 39
Integer (class in laurelin.Ldap.schema), 74
integerFirstComponentMatch (class in laurelin.Ldap.schema), 78
integerMatch (class in laurelin.Ldap.schema), 79
InvalidBindState, 57
InvalidSyntaxError, 57
iterattrs() (laurelin.Ldap.attrsdict.AttrsDict method), 38
- ## J
- JPEG (class in laurelin.Ldap.schema), 74
- ## K
- keyword (laurelin.Ldap.controls.Control attribute), 29

L

[laurelin.extensions.descattrs \(module\)](#), 17
[laurelin.extensions.pagedresults \(module\)](#), 19
[laurelin.ldap \(module\)](#), 80
[laurelin.ldap.attributetype \(module\)](#), 35
[laurelin.ldap.attrsdict \(module\)](#), 38
[laurelin.ldap.attrvaluelist \(module\)](#), 38
[laurelin.ldap.base \(module\)](#), 39
[laurelin.ldap.config \(module\)](#), 53
[laurelin.ldap.constants \(module\)](#), 55
[laurelin.ldap.exceptions \(module\)](#), 56
[laurelin.ldap.extensible \(module\)](#), 58
[laurelin.ldap.filter \(module\)](#), 58
[laurelin.ldap.ldapobject \(module\)](#), 59
[laurelin.ldap.modify \(module\)](#), 66
[laurelin.ldap.net \(module\)](#), 66
[laurelin.ldap.objectclass \(module\)](#), 68
[laurelin.ldap.rules \(module\)](#), 69
[laurelin.ldap.schema \(module\)](#), 71
[laurelin.ldap.validation \(module\)](#), 80
[LDAP \(class in laurelin.ldap\)](#), 80
[LDAP \(class in laurelin.ldap.base\)](#), 39
[LDAPConnectionError](#), 57
[LDAPError](#), 57, 94
[LDAPExtensionError](#), 57
[LDAPAPI_SOCKET_PATHS](#) ([laurelin.ldap.net.LDAPSocket](#) attribute), 67
[LDAPObject \(class in laurelin.ldap\)](#), 95
[LDAPObject \(class in laurelin.ldap.ldapobject\)](#), 59
[LDAPObject.add_desc_attrs\(\) \(in module laurelin.extensions.descattrs\)](#), 18
[LDAPObject.delete_desc_attrs\(\) \(in module laurelin.extensions.descattrs\)](#), 18
[LDAPObject.desc_attrs\(\) \(in module laurelin.extensions.descattrs\)](#), 18
[LDAPObject.replace_desc_attrs\(\) \(in module laurelin.extensions.descattrs\)](#), 18
[LDAPResponse \(class in laurelin.ldap.base\)](#), 52
[LDAPSASLError](#), 57
[LDAPSchemaError](#), 57
[LDAPSocket \(class in laurelin.ldap.net\)](#), 66
[LDAPSUnsupportedError](#), 57
[LDAPSyntaxDescription \(class in laurelin.ldap.schema\)](#), 74
[LDAPTransactionError](#), 57
[LDAPUnicodeWarning](#), 57
[LDAPUnsolicitedMessage](#), 57
[LDAPURI \(class in laurelin.ldap\)](#), 93
[LDAPURI \(class in laurelin.ldap.base\)](#), 52
[LDAPValidationError](#), 57
[LDAPWarning](#), 57
[load_config_dict\(\) \(in module laurelin.ldap.config\)](#), 54
[load_file\(\) \(in module laurelin.ldap.config\)](#), 54
[LOG_FORMAT \(laurelin.ldap.base.LDAP attribute\)](#), 42

[LOG_FORMAT \(laurelin.ldap.LDAP attribute\)](#), 83
[log_warnings\(\) \(laurelin.ldap.base.LDAP static method\)](#), 46
[log_warnings\(\) \(laurelin.ldap.LDAP static method\)](#), 87

M

[match\(\) \(laurelin.ldap.attributetype.DefaultMatchingRule method\)](#), 37
[match\(\) \(laurelin.ldap.rules.MatchingRule method\)](#), 70
[MatchingRule \(class in laurelin.ldap.rules\)](#), 69
[MatchingRuleDescription \(class in laurelin.ldap.schema\)](#), 74
[MatchingRuleUseDescription \(class in laurelin.ldap.schema\)](#), 75
[may \(laurelin.ldap.objectclass.ObjectClass attribute\)](#), 69
[MetaMatchingRule \(class in laurelin.ldap.rules\)](#), 70
[MetaSyntaxRule \(class in laurelin.ldap.rules\)](#), 70
[method \(laurelin.ldap.controls.Control attribute\)](#), 29
[Mod \(class in laurelin.ldap\)](#), 99
[Mod \(class in laurelin.ldap.modify\)](#), 66
[mod_dn\(\) \(laurelin.ldap.base.LDAP method\)](#), 46
[mod_dn\(\) \(laurelin.ldap.LDAP method\)](#), 87
[mod_dn\(\) \(laurelin.ldap.LDAPObject method\)](#), 97
[mod_dn\(\) \(laurelin.ldap.ldapobject.LDAPObject method\)](#), 61
[mod_dn\(\) \(laurelin.ldap.ldapobject.ModTransactionObject method\)](#), 65
[mod_transaction\(\) \(laurelin.ldap.LDAPObject method\)](#), 97
[mod_transaction\(\) \(laurelin.ldap.ldapobject.LDAPObject method\)](#), 62
[modify\(\) \(laurelin.ldap.base.LDAP method\)](#), 46
[modify\(\) \(laurelin.ldap.LDAP method\)](#), 87
[modify\(\) \(laurelin.ldap.LDAPObject method\)](#), 97
[modify\(\) \(laurelin.ldap.ldapobject.LDAPObject method\)](#), 62
[modify\(\) \(laurelin.ldap.ldapobject.ModTransactionObject method\)](#), 65
[Modlist\(\) \(in module laurelin.ldap.modify\)](#), 66
[ModTransactionObject \(class in laurelin.ldap.ldapobject\)](#), 64
[move\(\) \(laurelin.ldap.base.LDAP method\)](#), 47
[move\(\) \(laurelin.ldap.LDAP method\)](#), 87
[move\(\) \(laurelin.ldap.LDAPObject method\)](#), 98
[move\(\) \(laurelin.ldap.ldapobject.LDAPObject method\)](#), 62
[MultipleSearchResults](#), 58
[must \(laurelin.ldap.objectclass.ObjectClass attribute\)](#), 69

N

[NAME \(laurelin.ldap.rules.MatchingRule attribute\)](#), 70
[NAME \(laurelin.ldap.schema.bitStringMatch attribute\)](#), 77
[NAME \(laurelin.ldap.schema.booleanMatch attribute\)](#), 77

NAME (laurelin.Ldap.schema.caseExactIA5Match attribute), 77

NAME (laurelin.Ldap.schema.caseExactMatch attribute), 77

NAME (laurelin.Ldap.schema.caseIgnoreIA5Match attribute), 77

NAME (laurelin.Ldap.schema.caseIgnoreListMatch attribute), 78

NAME (laurelin.Ldap.schema.caseIgnoreMatch attribute), 78

NAME (laurelin.Ldap.schema.directoryStringFirstComponentMatch attribute), 78

NAME (laurelin.Ldap.schema.distinguishedNameMatch attribute), 78

NAME (laurelin.Ldap.schema.generalizedTimeMatch attribute), 78

NAME (laurelin.Ldap.schema.integerFirstComponentMatch attribute), 78

NAME (laurelin.Ldap.schema.integerMatch attribute), 79

NAME (laurelin.Ldap.schema.numericStringMatch attribute), 79

NAME (laurelin.Ldap.schema.objectIdentifierFirstComponentMatch attribute), 79

NAME (laurelin.Ldap.schema.objectIdentifierMatch attribute), 79

NAME (laurelin.Ldap.schema.octetStringMatch attribute), 79

NAME (laurelin.Ldap.schema.telephoneNumberMatch attribute), 79

NAME (laurelin.Ldap.schema.uniqueMemberMatch attribute), 79

NameAndOptionalUID (class in laurelin.Ldap.schema), 75

NameFormDescription (class in laurelin.Ldap.schema), 75

NEVER (laurelin.Ldap.constants.DerefAliases attribute), 56

NEVER (laurelin.Ldap.DerefAliases attribute), 94

NO_ATTRS (laurelin.Ldap.base.LDAP attribute), 42

NO_ATTRS (laurelin.Ldap.LDAP attribute), 83

normalize_global_config_param() (in module laurelin.Ldap.config), 55

NoSearchResults, 58, 94

NumericString (class in laurelin.Ldap.schema), 75

numericStringMatch (class in laurelin.Ldap.schema), 79

O

obj() (laurelin.Ldap.base.LDAP method), 47

obj() (laurelin.Ldap.LDAP method), 88

obj() (laurelin.Ldap.LDAPObject method), 98

obj() (laurelin.Ldap.Ldapobject.LDAPObject method), 62

ObjectClass (class in laurelin.Ldap.objectclass), 68

ObjectClassDescription (class in laurelin.Ldap.schema), 75

objectIdentifierFirstComponentMatch (class in laurelin.Ldap.schema), 79

objectIdentifierMatch (class in laurelin.Ldap.schema), 79

OctetString (class in laurelin.Ldap.schema), 75

octetStringMatch (class in laurelin.Ldap.schema), 79

OID (class in laurelin.Ldap.schema), 75

OID (laurelin.Ldap.rules.MatchingRule attribute), 70

OID (laurelin.Ldap.rules.SyntaxRule attribute), 70

OID (laurelin.Ldap.schema.AttributeTypeDescription attribute), 71

OID (laurelin.Ldap.schema.Binary attribute), 71

OID (laurelin.Ldap.schema.BitString attribute), 71

OID (laurelin.Ldap.schema.bitStringMatch attribute), 77

OID (laurelin.Ldap.schema.Boolean attribute), 71

OID (laurelin.Ldap.schema.booleanMatch attribute), 77

OID (laurelin.Ldap.schema.caseExactIA5Match attribute), 77

OID (laurelin.Ldap.schema.caseExactMatch attribute), 77

OID (laurelin.Ldap.schema.caseIgnoreIA5Match attribute), 78

OID (laurelin.Ldap.schema.caseIgnoreListMatch attribute), 78

OID (laurelin.Ldap.schema.caseIgnoreMatch attribute), 78

OID (laurelin.Ldap.schema.Certificate attribute), 72

OID (laurelin.Ldap.schema.CountryString attribute), 72

OID (laurelin.Ldap.schema.DeliveryMethod attribute), 72

OID (laurelin.Ldap.schema.DirectoryString attribute), 72

OID (laurelin.Ldap.schema.directoryStringFirstComponentMatch attribute), 78

OID (laurelin.Ldap.schema.DistinguishedName attribute), 73

OID (laurelin.Ldap.schema.distinguishedNameMatch attribute), 78

OID (laurelin.Ldap.schema.DITContentRuleDescription attribute), 72

OID (laurelin.Ldap.schema.DITStructureRuleDescription attribute), 72

OID (laurelin.Ldap.schema.EnhancedGuide attribute), 73

OID (laurelin.Ldap.schema.FacsimileTelephoneNumber attribute), 73

OID (laurelin.Ldap.schema.Fax attribute), 73

OID (laurelin.Ldap.schema.GeneralizedTime attribute), 73

OID (laurelin.Ldap.schema.generalizedTimeMatch attribute), 78

OID (laurelin.Ldap.schema.Guide attribute), 74

OID (laurelin.Ldap.schema.IA5String attribute), 74

OID (laurelin.Ldap.schema.Integer attribute), 74

OID (laurelin.Ldap.schema.integerFirstComponentMatch attribute), 78

OID (laurelin.Ldap.schema.integerMatch attribute), 79

OID (laurelin.Ldap.schema.JPEG attribute), 74

- OID (laurelin.Ldap.schema.LDAPSyntaxDescription attribute), 74
 - OID (laurelin.Ldap.schema.MatchingRuleDescription attribute), 75
 - OID (laurelin.Ldap.schema.MatchingRuleUseDescription attribute), 75
 - OID (laurelin.Ldap.schema.NameAndOptionalUID attribute), 75
 - OID (laurelin.Ldap.schema.NameFormDescription attribute), 75
 - OID (laurelin.Ldap.schema.NumericString attribute), 75
 - OID (laurelin.Ldap.schema.numericStringMatch attribute), 79
 - OID (laurelin.Ldap.schema.ObjectClassDescription attribute), 75
 - OID (laurelin.Ldap.schema.objectIdentifierFirstComponentMatch attribute), 79
 - OID (laurelin.Ldap.schema.objectIdentifierMatch attribute), 79
 - OID (laurelin.Ldap.schema.OctetString attribute), 75
 - OID (laurelin.Ldap.schema.octetStringMatch attribute), 79
 - OID (laurelin.Ldap.schema.OID attribute), 75
 - OID (laurelin.Ldap.schema.OtherMailbox attribute), 76
 - OID (laurelin.Ldap.schema.PostalAddress attribute), 76
 - OID (laurelin.Ldap.schema.PrintableString attribute), 76
 - OID (laurelin.Ldap.schema.SubstringAssertion attribute), 76
 - OID (laurelin.Ldap.schema.TelephoneNumber attribute), 76
 - OID (laurelin.Ldap.schema.telephoneNumberMatch attribute), 79
 - OID (laurelin.Ldap.schema.TeletextTerminalIdentifier attribute), 77
 - OID (laurelin.Ldap.schema.TelexNumber attribute), 77
 - OID (laurelin.Ldap.schema.uniqueMemberMatch attribute), 79
 - OID_DISCONNECTION_NOTICE (laurelin.Ldap.net.LDAPSocket attribute), 67
 - OID_OBJ_CLASS_ATTR (laurelin.Ldap.base.LDAP attribute), 42
 - OID_OBJ_CLASS_ATTR (laurelin.Ldap.LDAP attribute), 83
 - OID_STARTTLS (laurelin.Ldap.base.LDAP attribute), 42
 - OID_STARTTLS (laurelin.Ldap.LDAP attribute), 83
 - OID_WHOAMI (laurelin.Ldap.base.LDAP attribute), 42
 - OID_WHOAMI (laurelin.Ldap.LDAP attribute), 83
 - ONE (laurelin.Ldap.constants.Scope attribute), 56
 - ONE (laurelin.Ldap.Scope attribute), 93
 - ONELEVEL (laurelin.Ldap.constants.Scope attribute), 56
 - ONELEVEL (laurelin.Ldap.Scope attribute), 93
 - op_to_string() (laurelin.Ldap.Mod static method), 100
 - op_to_string() (laurelin.Ldap.modify.Mod static method), 66
 - optional (class in laurelin.Ldap), 28, 94
 - OtherMailbox (class in laurelin.Ldap.schema), 76
- ## P
- parse() (in module laurelin.Ldap.filter), 58
 - parse_simple_filter() (in module laurelin.Ldap.filter), 58
 - parse_standard_filter() (in module laurelin.Ldap.filter), 59
 - PostalAddress (class in laurelin.Ldap.schema), 76
 - prep_methods (laurelin.Ldap.rules.MatchingRule attribute), 70
 - prep_methods (laurelin.Ldap.schema.caseExactIA5Match attribute), 77
 - prep_methods (laurelin.Ldap.schema.caseExactMatch attribute), 77
 - prep_methods (laurelin.Ldap.schema.caseIgnoreIA5Match attribute), 78
 - prep_methods (laurelin.Ldap.schema.caseIgnoreListMatch attribute), 78
 - prep_methods (laurelin.Ldap.schema.caseIgnoreMatch attribute), 78
 - prep_methods (laurelin.Ldap.schema.numericStringMatch attribute), 79
 - prep_methods (laurelin.Ldap.schema.telephoneNumberMatch attribute), 79
 - prepare() (laurelin.Ldap.attributetype.DefaultMatchingRule method), 37
 - prepare() (laurelin.Ldap.controls.Control method), 29
 - prepare() (laurelin.Ldap.rules.MatchingRule method), 70
 - PrintableString (class in laurelin.Ldap.schema), 76
 - process_ldif() (laurelin.Ldap.base.LDAP method), 47
 - process_ldif() (laurelin.Ldap.LDAP method), 88
 - ProhibitedCharacterError, 58
- ## R
- rdn() (laurelin.Ldap.LDAPObject method), 98
 - rdn() (laurelin.Ldap.Ldapobject.LDAPObject method), 62
 - recheck_sasl_mechs() (laurelin.Ldap.base.LDAP method), 48
 - recheck_sasl_mechs() (laurelin.Ldap.LDAP method), 88
 - RECV_BUFFER (laurelin.Ldap.net.LDAPSocket attribute), 67
 - recv_messages() (laurelin.Ldap.net.LDAPSocket method), 67
 - recv_one() (laurelin.Ldap.net.LDAPSocket method), 67
 - recv_response() (laurelin.Ldap.base.ExtendedResponseHandle method), 39
 - refresh() (laurelin.Ldap.LDAPObject method), 98
 - refresh() (laurelin.Ldap.Ldapobject.LDAPObject method), 63
 - refresh_all() (laurelin.Ldap.LDAPObject method), 98
 - refresh_all() (laurelin.Ldap.Ldapobject.LDAPObject method), 63

- `refresh_missing()` (laurelin.Ldap.LDAPObject method), 98
 - `refresh_missing()` (laurelin.Ldap.Ldapobject.LDAPObject method), 63
 - `refresh_root_dse()` (laurelin.Ldap.base.LDAP method), 48
 - `refresh_root_dse()` (laurelin.Ldap.LDAP method), 88
 - `regex` (laurelin.Ldap.rules.RegexSyntaxRule attribute), 70
 - `regex` (laurelin.Ldap.schema.AttributeTypeDescription attribute), 71
 - `regex` (laurelin.Ldap.schema.BitString attribute), 71
 - `regex` (laurelin.Ldap.schema.CountryString attribute), 72
 - `regex` (laurelin.Ldap.schema.DeliveryMethod attribute), 72
 - `regex` (laurelin.Ldap.schema.DistinguishedName attribute), 73
 - `regex` (laurelin.Ldap.schema.DITContentRuleDescription attribute), 72
 - `regex` (laurelin.Ldap.schema.DITStructureRuleDescription attribute), 72
 - `regex` (laurelin.Ldap.schema.GeneralizedTime attribute), 73
 - `regex` (laurelin.Ldap.schema.IA5String attribute), 74
 - `regex` (laurelin.Ldap.schema.Integer attribute), 74
 - `regex` (laurelin.Ldap.schema.LDAPSyntaxDescription attribute), 74
 - `regex` (laurelin.Ldap.schema.MatchingRuleDescription attribute), 75
 - `regex` (laurelin.Ldap.schema.MatchingRuleUseDescription attribute), 75
 - `regex` (laurelin.Ldap.schema.NameAndOptionalUID attribute), 75
 - `regex` (laurelin.Ldap.schema.NameFormDescription attribute), 75
 - `regex` (laurelin.Ldap.schema.NumericString attribute), 75
 - `regex` (laurelin.Ldap.schema.ObjectClassDescription attribute), 75
 - `regex` (laurelin.Ldap.schema.OID attribute), 75
 - `regex` (laurelin.Ldap.schema.OtherMailbox attribute), 76
 - `regex` (laurelin.Ldap.schema.PostalAddress attribute), 76
 - `regex` (laurelin.Ldap.schema.PrintableString attribute), 76
 - `regex` (laurelin.Ldap.schema.SubstringAssertion attribute), 76
 - `regex` (laurelin.Ldap.schema.TeletextTerminalIdentifier attribute), 77
 - `regex` (laurelin.Ldap.schema.TelexNumber attribute), 77
 - `RegexSyntaxRule` (class in laurelin.Ldap.rules), 70
 - `remove()` (laurelin.Ldap.attrvaluelist.AttrValueList method), 39
 - `rename()` (laurelin.Ldap.base.LDAP method), 48
 - `rename()` (laurelin.Ldap.LDAP method), 89
 - `rename()` (laurelin.Ldap.LDAPObject method), 99
 - `rename()` (laurelin.Ldap.Ldapobject.LDAPObject method), 63
 - `REPLACE` (laurelin.Ldap.Mod attribute), 100
 - `REPLACE` (laurelin.Ldap.modify.Mod attribute), 66
 - `replace_attrs()` (laurelin.Ldap.base.LDAP method), 48
 - `replace_attrs()` (laurelin.Ldap.LDAP method), 89
 - `replace_attrs()` (laurelin.Ldap.LDAPObject method), 99
 - `replace_attrs()` (laurelin.Ldap.Ldapobject.LDAPObject method), 63
 - `REQUEST_OID` (laurelin.Ldap.controls.Control attribute), 29
 - `required_attr()` (laurelin.Ldap.objectclass.ObjectClass method), 69
 - `response_attr` (laurelin.Ldap.controls.Control attribute), 29
 - `RESPONSE_OID` (laurelin.Ldap.controls.Control attribute), 29
 - `ResponseHandle` (class in laurelin.Ldap.base), 52
 - `rfc4511_filter_to_rfc4515_string()` (in module laurelin.Ldap.filter), 59
- ## S
- `sasl_bind()` (laurelin.Ldap.base.LDAP method), 48
 - `sasl_bind()` (laurelin.Ldap.LDAP method), 89
 - `sasl_init()` (laurelin.Ldap.net.LDAPSocket method), 67
 - `sasl_mech` (laurelin.Ldap.net.LDAPSocket attribute), 67
 - `sasl_process_auth_challenge()` (laurelin.Ldap.net.LDAPSocket method), 67
 - `sasl_qop` (laurelin.Ldap.net.LDAPSocket attribute), 67
 - `SchemaValidator` (class in laurelin.Ldap.schema), 76
 - `Scope` (class in laurelin.Ldap), 93
 - `Scope` (class in laurelin.Ldap.constants), 56
 - `SEARCH` (laurelin.Ldap.constants.DerefAliases attribute), 56
 - `SEARCH` (laurelin.Ldap.DerefAliases attribute), 94
 - `search()` (laurelin.Ldap.base.LDAP method), 49
 - `search()` (laurelin.Ldap.base.LDAPURI method), 52
 - `search()` (laurelin.Ldap.LDAP method), 89
 - `search()` (laurelin.Ldap.LDAPObject method), 99
 - `search()` (laurelin.Ldap.Ldapobject.LDAPObject method), 63
 - `search()` (laurelin.Ldap.LDAPURI method), 93
 - `SearchReferenceHandle` (class in laurelin.Ldap.base), 53
 - `SearchResultHandle` (class in laurelin.Ldap.base), 53
 - `send_extended_request()` (laurelin.Ldap.base.LDAP method), 50
 - `send_extended_request()` (laurelin.Ldap.LDAP method), 91
 - `send_message()` (laurelin.Ldap.net.LDAPSocket method), 67
 - `set_global_config()` (in module laurelin.Ldap.config), 55
 - `setdefault()` (laurelin.Ldap.attrsdict.AttrsDict method), 38
 - `SIMPLE` (laurelin.Ldap.constants.FilterSyntax attribute), 56
 - `SIMPLE` (laurelin.Ldap.FilterSyntax attribute), 94
 - `simple_bind()` (laurelin.Ldap.base.LDAP method), 50
 - `simple_bind()` (laurelin.Ldap.LDAP method), 91

- STANDARD (laurelin.ldap.constants.FilterSyntax attribute), 56
- STANDARD (laurelin.ldap.FilterSyntax attribute), 94
- start_tls() (laurelin.ldap.base.LDAP method), 50
- start_tls() (laurelin.ldap.LDAP method), 91
- start_tls() (laurelin.ldap.net.LDAPSocket method), 68
- string() (laurelin.ldap.constants.FilterSyntax static method), 56
- string() (laurelin.ldap.constants.Scope static method), 56
- string() (laurelin.ldap.FilterSyntax static method), 94
- string() (laurelin.ldap.Mod static method), 100
- string() (laurelin.ldap.modify.Mod static method), 66
- string() (laurelin.ldap.Scope static method), 94
- SUB (laurelin.ldap.constants.Scope attribute), 56
- SUB (laurelin.ldap.Scope attribute), 93
- SubstringAssertion (class in laurelin.ldap.schema), 76
- SUBTREE (laurelin.ldap.constants.Scope attribute), 56
- SUBTREE (laurelin.ldap.Scope attribute), 94
- syntax (laurelin.ldap.attributetype.AttributeType attribute), 36
- syntax (laurelin.ldap.attributetype.DefaultAttributeType attribute), 37
- SYNTAX (laurelin.ldap.rules.MatchingRule attribute), 70
- SYNTAX (laurelin.ldap.schema.bitStringMatch attribute), 77
- SYNTAX (laurelin.ldap.schema.booleanMatch attribute), 77
- SYNTAX (laurelin.ldap.schema.caseExactIA5Match attribute), 77
- SYNTAX (laurelin.ldap.schema.caseExactMatch attribute), 77
- SYNTAX (laurelin.ldap.schema.caseIgnoreIA5Match attribute), 78
- SYNTAX (laurelin.ldap.schema.caseIgnoreListMatch attribute), 78
- SYNTAX (laurelin.ldap.schema.caseIgnoreMatch attribute), 78
- SYNTAX (laurelin.ldap.schema.directoryStringFirstComponentMatch attribute), 78
- SYNTAX (laurelin.ldap.schema.distinguishedNameMatch attribute), 78
- SYNTAX (laurelin.ldap.schema.generalizedTimeMatch attribute), 78
- SYNTAX (laurelin.ldap.schema.integerFirstComponentMatch attribute), 78
- SYNTAX (laurelin.ldap.schema.integerMatch attribute), 79
- SYNTAX (laurelin.ldap.schema.numericStringMatch attribute), 79
- SYNTAX (laurelin.ldap.schema.objectIdentifierFirstComponentMatch attribute), 79
- SYNTAX (laurelin.ldap.schema.objectIdentifierMatch attribute), 79
- SYNTAX (laurelin.ldap.schema.octetStringMatch attribute), 79
- SYNTAX (laurelin.ldap.schema.telephoneNumberMatch attribute), 79
- SYNTAX (laurelin.ldap.schema.uniqueMemberMatch attribute), 79
- SyntaxRule (class in laurelin.ldap.rules), 70
- ## T
- tag() (laurelin.ldap.base.LDAP method), 51
- tag() (laurelin.ldap.LDAP method), 92
- TagError, 58
- TelephoneNumber (class in laurelin.ldap.schema), 76
- telephoneNumberMatch (class in laurelin.ldap.schema), 79
- TeletextTerminalIdentifier (class in laurelin.ldap.schema), 77
- TelexNumber (class in laurelin.ldap.schema), 77
- ## U
- unbind() (laurelin.ldap.base.LDAP method), 51
- unbind() (laurelin.ldap.LDAP method), 92
- UnexpectedResponseType, 58
- UnexpectedSearchResults, 58
- UNIFIED (laurelin.ldap.constants.FilterSyntax attribute), 56
- UNIFIED (laurelin.ldap.FilterSyntax attribute), 94
- uniqueMemberMatch (class in laurelin.ldap.schema), 79
- update() (laurelin.ldap.attrsdict.AttrsDict method), 38
- ## V
- validate() (laurelin.ldap.attributetype.AttributeType method), 36
- validate() (laurelin.ldap.attributetype.DefaultMatchingRule method), 37
- validate() (laurelin.ldap.attributetype.DefaultSyntaxRule method), 37
- validate() (laurelin.ldap.attrsdict.AttrsDict static method), 38
- validate() (laurelin.ldap.LDAPObject method), 99
- validate() (laurelin.ldap.ldapobject.LDAPObject method), 64
- validate() (laurelin.ldap.rules.MatchingRule method), 70
- validate() (laurelin.ldap.rules.RegexSyntaxRule method), 70
- validate() (laurelin.ldap.rules.SyntaxRule method), 71
- validate() (laurelin.ldap.schema.Binary method), 71
- validate() (laurelin.ldap.schema.Boolean method), 71
- validate() (laurelin.ldap.schema.Certificate method), 72
- validate() (laurelin.ldap.schema.DirectoryString method), 72
- validate() (laurelin.ldap.schema.EnhancedGuide method), 73
- validate() (laurelin.ldap.schema.FacsimileTelephoneNumber method), 73

`validate()` (`laurelin.ldap.schema.Fax` method), [73](#)
`validate()` (`laurelin.ldap.schema.GeneralizedTime` method), [73](#)
`validate()` (`laurelin.ldap.schema.Guide` method), [74](#)
`validate()` (`laurelin.ldap.schema.JPEG` method), [74](#)
`validate()` (`laurelin.ldap.schema.OctetString` method), [75](#)
`validate()` (`laurelin.ldap.schema.TelephoneNumber` method), [76](#)
`validate_attr()` (`laurelin.ldap.attrsdict.AttrsDict` static method), [38](#)
`validate_modify()` (`laurelin.ldap.base.LDAP` method), [51](#)
`validate_modify()` (`laurelin.ldap.LDAP` method), [92](#)
`validate_modify()` (`laurelin.ldap.LDAPObject` method), [99](#)
`validate_modify()` (`laurelin.ldap.ldapobject.LDAPObject` method), [64](#)
`validate_modify()` (`laurelin.ldap.validation.Validator` method), [80](#)
`validate_object()` (`laurelin.ldap.base.LDAP` method), [51](#)
`validate_object()` (`laurelin.ldap.LDAP` method), [92](#)
`validate_object()` (`laurelin.ldap.schema.SchemaValidator` method), [76](#)
`validate_object()` (`laurelin.ldap.validation.Validator` method), [80](#)
`validate_values()` (`laurelin.ldap.attrsdict.AttrsDict` static method), [38](#)
`Validator` (class in `laurelin.ldap.validation`), [80](#)

W

`who_am_i()` (`laurelin.ldap.base.LDAP` method), [52](#)
`who_am_i()` (`laurelin.ldap.LDAP` method), [92](#)