python-Idap Documentation

Release 3.0.0

python-Idap project

March 27, 2018
What is python-ldap?

python-ldap provides an object-oriented API to access LDAP directory servers from Python programs. For LDAP operations the module wraps OpenLDAP’s client library, libldap. Additionally, the package contains modules for other LDAP-related stuff:

- LDIF parsing and generation
- LDAP URLs
- LDAPv3 subschema
Installation instructions are available for several platforms.

Source code can be obtained using Git:

```
git clone https://github.com/python-ldap/python-ldap
```
Discussion about the use and future of python-ldap occurs in the python-ldap@python.org mailing list.
You can subscribe or unsubscribe to this list or browse the list archive.
The documentation for python-ldap 3.x is hosted at Read the Docs.

You can switch between versions of the library, or download PDF or HTML versions for offline use, using the sidebar on the right.

Documentation for some older versions is available for download at the GitHub release page.
5.1 Installing python-ldap

5.1.1 Installing from PyPI

The preferred point for downloading the “official” source distribution is the PyPI repository which supports installing via pip. For example:

```
$ python -m pip install python-ldap
```

For installing from PyPI, you will need the same Build prerequisites as when installing from source.

We do not currently provide pre-built packages (wheels).

Furthermore, python-ldap requires the modules pyasn1 and pyasn1-modules. pip will install these automatically.

5.1.2 Pre-built Binaries

Because distributions seem to be all over the place, this page tries to list all the current ones we know of.

Note that the python-ldap team is not responsible for the binary packages except the sources you can grab from the PyPI page. Also note that binary packages are most times not up to date. If you experience troubles with a binary package, it would be nice if you try to build a recent version of python-ldap before submitting a bug report to make sure you did not hit a problem already fixed in recent releases.

**openSUSE Linux**

Ships with python-ldap and there’s an additional download repository which contains builds of latest releases (see also OBS package).
Debian Linux

Have a look into the Debian Package Tracker to get up to date information which versions are available.

Windows

Unofficial packages for Windows are available on Christoph Gohlke’s page.

FreeBSD

The CVS repository of FreeBSD contains the package py-ldap

Mac OS X

You can install directly with pip:

```
$ xcode-select --install
$ pip install python-ldap \
   --global-option=build_ext \
   --global-option="-I$(/usr/include/sasl"
```

5.1.3 Installing from Source

python-ldap is built and installed using the Python setuptools. From a source repository:

```
$ python -m pip install setuptools
$ python setup.py install
```

If you have more than one Python interpreter installed locally, you should use the same one you plan to use python-ldap with.

Further instructions can be found in Setuptools documentation.

5.1.4 Build prerequisites

The following software packages are required to be installed on the local system when building python-ldap:

- Python version 2.7, or 3.4 or later including its development files
- C compiler corresponding to your Python version (on Linux, it is usually gcc)
- OpenLDAP client libs version 2.4.11 or later; it is not possible and not supported to build with prior versions.
- OpenSSL (optional)
- Cyrus SASL (optional)
- Kerberos libraries, MIT or Heimdal (optional)
Alpine

Packages for building:

```bash
# apk add build-base openldap-dev python2-dev python3-dev
```

CentOS

Packages for building:

```bash
# yum groupinstall "Development tools"
# yum install openldap-devel python-devel
```

Debian

Packages for building and testing:

```bash
# apt-get install build-essential python3-dev python2.7-dev \
   libldap2-dev libsasl2-dev slapd ldap-utils python-tox \
   lcov valgrind
```

Fedora

Packages for building and testing:

```bash
# dnf install "@C Development Tools and Libraries" openldap-devel \
   python2-devel python3-devel python3-tox \
   lcov clang-analyzer valgrind
```

Note: openldap-2.4.45-2 (Fedora 26), openldap-2.4.45-4 (Fedora 27) or newer are required.

5.1.5 setup.cfg

The file setup.cfg allows to set some build and installation parameters for reflecting the local installation of required software packages. Only section `[_ldap]` is described here. More information about other sections can be found in Setuptools documentation.

`library_dirs`

Specifies in which directories to search for required libraries.

`include_dirs`

Specifies in which directories to search for include files of required libraries.

`libs`

A space-separated list of library names to link to (see Libraries used).

`extra_compile_args`

Compiler options.

`extra_objects`

5.1. Installing python-ldap
Libraries used

ldap

ldap_r
The LDAP protocol library of OpenLDAP. ldap_r is the reentrant version and should be preferred.

lber
The BER encoder/decoder library of OpenLDAP.

sasl2
The Cyrus-SASL library (optional)

ssl
The SSL/TLS library of OpenSSL (optional)

crypto
The basic cryptographic library of OpenSSL (optional)

Example

The following example is for a full-featured build (including SSL and SASL support) of python-ldap with OpenLDAP installed in a different prefix directory (here /opt/openldap-2.4) and SASL header files found in /usr/include/sasl. Debugging symbols are preserved with compile option -g.

```python
[._ldap]
library_dirs = /opt/openldap-2.4/lib
include_dirs = /opt/openldap-2.4/include /usr/include/sasl
extra_compile_args = -g
extra_objects =
libs = ldap_r lber sasl2 ssl crypto
```

5.2 Bytes/text management

Python 3 introduces a hard distinction between text (str) – sequences of characters (formally, Unicode codepoints) – and bytes – sequences of 8-bit values used to encode any kind of data for storage or transmission.

Python 2 has the same distinction between str (bytes) and unicode (text). However, values can be implicitly converted between these types as needed, e.g. when comparing or writing to disk or the network. The implicit encoding and decoding can be a source of subtle bugs when not designed and tested adequately.

In python-ldap 2.x (for Python 2), bytes were used for all fields, including those guaranteed to be text.

From version 3.0, python-ldap uses text where appropriate. On Python 2, the bytes mode setting influences how text is handled.

5.2.1 What’s text, and what’s bytes

The LDAP protocol states that some fields (distinguished names, relative distinguished names, attribute names, queries) be encoded in UTF-8. In python-ldap, these are represented as text (str on Python 3, unicode on Python 2).

Attribute values, on the other hand, MAY contain any type of data, including text. To know what type of data is represented, python-ldap would need access to the schema, which is not always available (nor always correct). Thus,
attribute values are always treated as bytes. Encoding/decoding to other formats — text, images, etc. — is left to the caller.

5.2.2 The bytes mode

In Python 3, text values are represented as str, the Unicode text type.

In Python 2, the behavior of python-ldap 3.0 is influenced by a bytes_mode argument to ldap.initialize():

bytes_mode=True (backwards compatible): Text values are represented as bytes (str) encoded using UTF-8.

bytes_mode=False (future compatible): Text values are represented as unicode.

If not given explicitly, python-ldap will default to bytes_mode=True, but if an unicode value supplied to it, if will warn and use that value.

Backwards-compatible behavior is not scheduled for removal until Python 2 itself reaches end of life.

5.2.3 Errors, warnings, and automatic encoding

While the type of values returned from python-ldap is always given by bytes_mode, for Python 2 the behavior for “wrong-type” values passed in can be controlled by the bytes_strictness argument to ldap.initialize():

bytes_strictness='error' (default if bytes_mode is specified): A TypeError is raised.

bytes_strictness='warn' (default when bytes_mode is not given explicitly): A warning is raised, and the value is encoded/decoded using the UTF-8 encoding.

The warnings are of type LDAPBytesWarning, which is a subclass of BytesWarning designed to be easily filtered out if needed.

bytes_strictness='silent': The value is automatically encoded/decoded using the UTF-8 encoding.

On Python 3, bytes_strictness is ignored and a TypeError is always raised.

When setting bytes_strictness, an explicit value for bytes_mode needs to be given as well.

5.2.4 Porting recommendations

Since end of life of Python 2 is coming in a few years, projects are strongly urged to make their code compatible with Python 3. General instructions for this are provided in Python documentation and in the Conservative porting guide.

When porting from python-ldap 2.x, users are advised to update their code to set bytes_mode=False, and fix any resulting failures.

The typical usage is as follows. Note that only the result’s values are of the bytes type:

```python
>>> import ldap
>>> con = ldap.initialize('ldap://localhost:389', bytes_mode=False)
>>> con.simple_bind_s(u'login', u'secret_password')
>>> results = con.search_s(u'ou=people,dc=example,dc=org', ldap.SCOPE_SUBTREE, u"(cn=Raphaël)")
>>> results
[('cn=Raphaël,ou=people,dc=example,dc=org", {
 'cn': [b'Raphaël']},
 'sn': [b'Barrois']},
```
5.2.5 Filtering warnings

The bytes mode warnings can be filtered out and ignored with a simple filter.

```python
import warnings
import ldap

if hasattr(ldap, 'LDAPBytesWarning'):
    warnings.simplefilter('ignore', ldap.LDAPBytesWarning)
```

5.3 python-ldap Reference Documentation

This document describes the package python-ldap with its various modules.

Depending on what you want to do this manual assumes basic to expert knowledge about the Python language and the LDAP standard (LDAPv3).

5.3.1 ldap LDAP library interface module

This module provides access to the LDAP (Lightweight Directory Access Protocol) C API implemented in OpenLDAP. It is similar to the C API, with the notable differences that lists are manipulated via Python list operations and errors appear as exceptions.

See also:

For more detailed information on the C interface, please see the (expired) draft-ietf-ldapext-ldap-c-api

This documentation is current for the Python LDAP module, version 3.0.0. Source and binaries are available from https://www.python-ldap.org/.

Functions

This module defines the following functions:

```python
ldap.initialize(uri[, trace_level=0[, trace_file=sys.stdout[, trace_stack_limit=None[, bytes_mode=None[, bytes_strictness=None ]]]]]) → LDAPObject object
```

Initializes a new connection object for accessing the given LDAP server, and return an LDAP object (see `LDAPObject classes`) used to perform operations on that server.

The `uri` parameter may be a comma- or whitespace-separated list of URIs containing only the schema, the host, and the port fields. Note that when using multiple URIs you cannot determine to which URI your client gets connected.

Note that internally the OpenLDAP function `ldap_initialize(3)` is called which just initializes the LDAP connection struct in the C API - nothing else. Therefore the first call to an operation method (bind, search etc.) then really opens the connection (lazy connect). Before that nothing is sent on the wire. The error handling in the calling application has to correctly handle this behaviour.
Three optional arguments are for generating debug log information: `trace_level` specifies the amount of information being logged, `trace_file` specifies a file-like object as target of the debug log and `trace_stack_limit` specifies the stack limit of tracebacks in debug log.

The `bytes_mode` and `bytes_strictness` arguments specify text/bytes behavior under Python 2. See `Bytes/text management` for a complete documentation.

Possible values for `trace_level` are 0 for no logging, 1 for only logging the method calls with arguments, 2 for logging the method calls with arguments and the complete results and 9 for also logging the traceback of method calls.

**See also:**


```python
ldap.get_option(option) → int|string
This function returns the value of the global option specified by `option`.
```

```python
ldap.set_option(option, invalue) → None
This function sets the value of the global option specified by `option` to `invalue`.
```

Changed in version 3.1: The deprecated functions `ldap.init()` and `ldap.open()` were removed.

**Constants**

The module defines various constants. Note that some constants depend on the build options and which underlying libs were used or even on the version of the libs. So before using those constants the application has to explicitly check whether they are available.

**General**

```python
ldap.PORT
The assigned TCP port number (389) that LDAP servers listen on.
```

```python
ldap.SASL_AVAIL
Integer where a non-zero value indicates that python-ldap was built with support for SASL (Cyrus-SASL).
```

```python
ldap.TLS_AVAIL
Integer where a non-zero value indicates that python-ldap was built with support for SSL/TLS (OpenSSL or similar libs).
```

**Options**

**See also:**

`ldap.conf(5)` and `ldap.get_option(3)`

For use with functions :py:func:`set_option()` and :py:func:`get_option()` and methods :py:method:`LDAPObject.set_option()` and :py:method:`LDAPObject.get_option()` the following option identifiers are defined as constants:

```python
ldap.OPT_API_FEATURE_INFO
```

```python
ldap.OPT_API_INFO
```

```python
ldap.OPT_CLIENT_CONTROLS
```

```python
ldap.OPT_DEBUG_LEVEL
Sets the debug level within the underlying OpenLDAP C lib (libldap). libldap sends the log messages to stderr.
```
ldap.OPT_DEFBASE

ldap.OPT_DEREF
   Specifies how alias dereferencing is done within the underlying LDAP C lib.

ldap.OPT_ERROR_STRING

ldap.OPT_DIAGNOSTIC_MESSAGE

ldap.OPT_HOST_NAME

ldap.OPT_MATCHED_DN

ldap.OPT_NETWORK_TIMEOUT
   Changed in version 3.0: A timeout of −1 or None resets timeout to infinity.

ldap.OPT_PROTOCOL_VERSION
   Sets the LDAP protocol version used for a connection. This is mapped to object attribute
   ldap.LDAPObject.protocol_version

ldap.OPT_REFERRALS
   int specifying whether referrals should be automatically chased within the underlying LDAP C lib.

ldap.OPT_REFHOPLIMIT

ldap.OPT_RESTART

ldap.OPT_SERVER_CONTROLS

ldap.OPT_SIZELIMIT

ldap.OPT_SUCCESS

ldap.OPT_TIMELIMIT

ldap.OPT_TIMEOUT
   Changed in version 3.0: A timeout of −1 or None resets timeout to infinity.

ldap.OPT_URI

SASL options

ldap.OPT_X_SASL_AUTHCID

ldap.OPT_X_SASL_AUTHZID

ldap.OPT_X_SASL_MECH

ldap.OPT_X_SASL_NOCANON
   If set to zero SASL host name canonicalization is disabled.

ldap.OPT_X_SASL_REALM

ldap.OPT_X_SASL_SECPROPS

ldap.OPT_X_SASL_SSF

ldap.OPT_X_SASL_SSF_EXTERNAL

ldap.OPT_X_SASL_SSF_MAX

ldap.OPT_X_SASL_SSF_MIN
TLS options

ldap.OPT_X_TLS
ldap.OPT_X_TLS_ALLOW
ldap.OPT_X_TLS_CACERTDIR
ldap.OPT_X_TLS_CACERTFILE
ldap.OPT_X_TLS_CERTFILE
ldap.OPT_X_TLS_CIPHER_SUITE
ldap.OPT_X_TLS_CTX
ldap.OPT_X_TLS_DEMAND
ldap.OPT_X_TLS_HARD
ldap.OPT_X_TLS_KEYFILE
ldap.OPT_X_TLS_NEVER
ldap.OPT_X_TLS_RANDOM_FILE
ldap.OPT_X_TLS_REQUIRE_CERT
ldap.OPT_X_TLS_TRY

Keepalive options

ldap.OPT_X_KEEPALIVE_IDLE
ldap.OPT_X_KEEPALIVE_PROBES
ldap.OPT_X_KEEPALIVE_INTERVAL

DN format flags

This constants are used for DN-parsing functions found in sub-module ldap.dn.

See also:
ldap_str2dn(3)
ldap.DN_FORMAT_LDAP
ldap.DN_FORMAT_LDAPV3
ldap.DN_FORMAT_LDAPV2
ldap.DN_FORMAT_DCE
ldap.DN_FORMAT_UFN
ldap.DN_FORMAT_AD_CANONICAL
ldap.DN_FORMAT_MASK
ldap.DN_PRETTY
ldap.DN_SKIP
ldap.DN_P_NOLEADTRAILSPACES
Exceptions

The module defines the following exceptions:

**exception ldap.LDAPError**
This is the base class of all exceptions raised by the module `ldap`. Unlike the C interface, errors are not returned as result codes, but are instead turned into exceptions, raised as soon as an the error condition is detected.

The exceptions are accompanied by a dictionary possibly containing an string value for the key `desc` (giving an English description of the error class) and/or a string value for the key `info` (giving a string containing more information that the server may have sent).

A third possible field of this dictionary is `matched` and is set to a truncated form of the name provided or alias dereferenced for the lowest entry (object or alias) that was matched.

**exception ldap.ADMINLIMIT_EXCEEDED**
**exception ldap.AFFECTS_MULTIPLE_DSAS**
**exception ldap.ALIAS_DEREF_PROBLEM**
A problem was encountered when dereferencing an alias. (Sets the `matched` field.)

**exception ldap.ALIAS_PROBLEM**
An alias in the directory points to a nonexistent entry. (Sets the `matched` field.)

**exception ldap.ALREADY_EXISTS**
The entry already exists. E.g. the `dn` specified with `add()` already exists in the DIT.

**exception ldap.AUTH_UNKNOWN**
The authentication method specified to `bind()` is not known.

**exception ldap.BUSY**
The DSA is busy.

**exception ldap.CLIENT_LOOP**
**exception ldap.COMPAR_FALSE**
A compare operation returned false. (This exception should never be seen because `compare()` returns a boolean result.)

**exception ldap.COMPAR_TRUE**
A compare operation returned true. (This exception should never be seen because `compare()` returns a boolean result.)

**exception ldap.CONFIDENTIALITY_REQUIRED**
Indicates that the session is not protected by a protocol such as Transport Layer Security (TLS), which provides session confidentiality.

**exception ldap.CONNECT_ERROR**
**exception ldap.CONSTRAINT_VIOLATION**
An attribute value specified or an operation started violates some server-side constraint (e.g., a postalAddress has too many lines or a line that is too long or a password is expired).

**exception ldap.CONTROL_NOT_FOUND**
**exception ldap.DECODING_ERROR**
An error was encountered decoding a result from the LDAP server.
exception ldap.ENCODING_ERROR
   An error was encountered encoding parameters to send to the LDAP server.

exception ldap.FILTER_ERROR
   An invalid filter was supplied to search() (e.g. unbalanced parentheses).

exception ldap.INAPPROPRIATE_AUTH
   Inappropriate authentication was specified (e.g. AUTH_SIMPLE was specified and the entry does not have a userPassword attribute).

exception ldap.INAPPROPRIATE_MATCHING
   Filter type not supported for the specified attribute.

exception ldap.INSUFFICIENT_ACCESS
   The user has insufficient access to perform the operation.

exception ldap.INVALID_CREDENTIALS
   Invalid credentials were presented during bind() or simple_bind(). (e.g., the wrong password).

exception ldap.INVALID_DN_SYNTAX
   A syntactically invalid DN was specified. (Sets the matched field.)

exception ldap.INVALID_SYNTAX
   An attribute value specified by the client did not comply to the syntax defined in the server-side schema.

exception ldap.IS_LEAF
   The object specified is a leaf of the directory tree. Sets the matched field of the exception dictionary value.

exception ldap.LOCAL_ERROR
   Some local error occurred. This is usually due to failed memory allocation.

exception ldapLOOP_DETECT
   A loop was detected.

exception ldap.MORE_RESULTS_TO_RETURN

exception ldap.NAMING_VIOLATION
   A naming violation occurred. This is raised e.g. if the LDAP server has constraints about the tree naming.

exception ldap.NO_OBJECT_CLASS_MODS
   Modifying the objectClass attribute as requested is not allowed (e.g. modifying structural object class of existing entry).

exception ldap.NOT_ALLOWED_ON_NONLEAF
   The operation is not allowed on a non-leaf object.

exception ldap.NOT_ALLOWED_ON_RDN
   The operation is not allowed on an RDN.

exception ldap.NOT_SUPPORTED

exception ldap.NO_MEMORY

exception ldap.NO_OBJECT_CLASS_MODS
   Object class modifications are not allowed.

exception ldap.NO_RESULTS_RETURNED

exception ldap.NO_SUCH_ATTRIBUTE
   The attribute type specified does not exist in the entry.

exception ldap.NO_SUCH_OBJECT
   The specified object does not exist in the directory. Sets the matched field of the exception dictionary value.
exception `ldap.OBJECT_CLASS_VIOLATION`
An object class violation occurred when the LDAP server checked the data sent by the client against the server-side schema (e.g. a "must" attribute was missing in the entry data).

exception `ldap.OPERATIONS_ERROR`
An operations error occurred.

exception `ldap.OTHER`
An unclassified error occurred.

exception `ldap.PARAM_ERROR`
An ldap routine was called with a bad parameter.

exception `ldap.PARTIAL_RESULTS`
Partial results only returned. This exception is raised if a referral is received when using LDAPv2. (This exception should never be seen with LDAPv3.)

exception `ldap.PROTOCOL_ERROR`
A violation of the LDAP protocol was detected.

exception `ldap.RESULTS_TOO_LARGE`
The result does not fit into a UDP packet. This happens only when using UDP-based CLDAP (connection-less LDAP) which is not supported anyway.

exception `ldap.SASL_BIND_IN_PROGRESS`

exception `ldap.SERVER_DOWN`
The LDAP library can't contact the LDAP server.

exception `ldap.SIZELIMIT_EXCEEDED`
An LDAP size limit was exceeded. This could be due to a sizelimit configuration on the LDAP server.

exception `ldap.STRONG_AUTH_NOT_SUPPORTED`
The LDAP server does not support strong authentication.

exception `ldap.STRONG_AUTH_REQUIRED`
Strong authentication is required for the operation.

exception `ldap.TIMELIMIT_EXCEEDED`
An LDAP time limit was exceeded.

exception `ldap.TIMEOUT`
A timelimit was exceeded while waiting for a result from the server.

exception `ldap.TYPE_OR_VALUE_EXISTS`
An attribute type or attribute value specified already exists in the entry.

exception `ldap.UNAVAILABLE`
The DSA is unavailable.

exception `ldap.UNAVAILABLE_CRITICAL_EXTENSION`
Indicates that the LDAP server was unable to satisfy a request because one or more critical extensions were not available. Either the server does not support the control or the control is not appropriate for the operation type.

exception `ldap.UNDEFINED_TYPE`
An attribute type used is not defined in the server-side schema.

exception `ldap.UNWILLING_TO_PERFORM`
The DSA is unwilling to perform the operation.

exception `ldap.USER_CANCELLED`
The operation was cancelled via the abandon() method.

The above exceptions are raised when a result code from an underlying API call does not indicate success.
Warnings

class ldap.LDAPBytesWarning
  Raised when bytes/text mismatch in non-strict bytes mode.

  See The bytes mode for details.

  New in version 3.0.0.

LDAPObject classes

class ldap.LDAPObject
  Instances of LDAPObject are returned by initialize() and open() (deprecated). The connection is automatically unbound and closed when the LDAP object is deleted.

  Internally LDAPObject is set to SimpleLDAPObject by default.

class ldap.SimpleLDAPObject (uri[, trace_level=0[, trace_file=sys.stdout[, trace_stack_limit=5]]])
  This basic class wraps all methods of the underlying C API object.

  The arguments are same like for function initialize().

class ldap.ReconnectLDAPObject (uri [, trace_level=0 [, trace_file=sys.stdout [, trace_stack_limit=5 [, retry_max=1 [, retry_delay=60.0]]]]])
  This class is derived from SimpleLDAPObject and used for automatic reconnects when using the synchronous request methods (see below). This class also implements the pickle protocol.

  The first arguments are same like for function initialize().

  For automatic reconnects it has additional arguments:

  * retry_max specifies the number of reconnect attempts before re-raising the ldap.SERVER_DOWN exception.
  * retry_delay specifies the time in seconds between reconnect attempts.

Arguments for LDAPv3 controls

The ldap.controls module can be used for constructing and decoding LDAPv3 controls. These arguments are available in the methods with names ending in _ext or _ext_s:

serverctrls is a list of ldap.controls.LDAPControl instances sent to the server along with the LDAP request (see module ldap.controls). These are controls which alter the behaviour of the server when processing the request if the control is supported by the server. The effect of controls might differ depending on the type of LDAP request or controls might not be applicable with certain LDAP requests at all.

clientctrls is a list of ldap.controls.LDAPControl instances passed to the client API and alter the behaviour of the client when processing the request.

Sending LDAP requests

Most methods on LDAP objects initiate an asynchronous request to the LDAP server and return a message id that can be used later to retrieve the result with result().

Methods with names ending in _s are the synchronous form and wait for and return with the server’s result, or with None if no data is expected.

LDAPObject instances have the following methods:
LDAPObject.abandon (msgid) → None

Abandons an LDAP operation in progress without waiting for a LDAP response. The msgid argument should be the message ID of an outstanding LDAP operation as returned by the asynchronous methods search(), modify(), etc. The caller can expect that the result of an abandoned operation will not be returned from a future call to result().

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

LDAPObject.add (dn, modlist) → int

Performs an LDAP add operation. The dn argument is the distinguished name (DN) of the entry to add, and modlist is a list of attributes to be added. The modlist is similar the one passed to modify(), except that the operation integer is omitted from the tuples in modlist. You might want to look into sub-module refmodule[ldap.modlist] for generating the modlist.

The asynchronous methods add() and add_ext() return the message ID of the initiated request.

The dn argument, and mod_type (second item) of modlist are text strings; see The bytes mode.

LDAPObject.bind (who, cred, method) → int

Send cancels extended operation for an LDAP operation specified by cancelid. The cancelid should be the message id of an outstanding LDAP operation as returned by the asynchronous methods search(), modify() etc. The caller can expect that the result of an abandoned operation will not be returned from a future call to result(). In opposite to abandon() this extended operation gets an result from the server and thus should be preferred if the server supports it.

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.


LDAPObject.compare (dn, attr, value) → int

The asynchronous technique yields the answer by raising the exception objects ldap.COMPARE_TRUE or ldap.COMPARE_FALSE.

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

The dn and attr arguments are text strings; see The bytes mode.

Note: A design fault in the LDAP API prevents value from containing NULL characters.
LDAPObject.delete(dn) → int
LDAPObject.delete_s(dn) → None
LDAPObject.delete_ext(dn[, serverctrls=None[, clientctrls=None]]) → int
LDAPObject.delete_ext_s(dn[, serverctrls=None[, clientctrls=None]]) → tuple

Performs an LDAP delete operation on dn. The asynchronous form returns the message id of the initiated request, and the result can be obtained from a subsequent call to result().

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

The dn argument is text string; see The bytes mode.

LDAPObject.extop(extreq[, serverctrls=None[, clientctrls=None]]) → int
LDAPObject.extop_s(extreq[, serverctrls=None[, clientctrls=None[, extop_resp_class=None]]]) → (respid, respvalue)

Performs an LDAP extended operation. The asynchronous form returns the message id of the initiated request, and the result can be obtained from a subsequent call to extop_result().

The extreq is an instance of class ldap.extop.ExtendedRequest containing the parameters for the extended operation request.

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

If argument extop_resp_class is set to a sub-class of ldap.extop.ExtendedResponse this class is used to return an object of this class instead of a raw BER value in respvalue.

LDAPObject.extop_result(self, msgid=ldap.RES_ANY, all=1, timeout=None) → (respid, respvalue)

Wrapper method around result4() just for retrieving the result of an extended operation sent before.

LDAPObject.modify(dn, modlist) → int
LDAPObject.modify_s(dn, modlist) → None
LDAPObject.modify_ext(dn, modlist[, serverctrls=None[, clientctrls=None]]) → int
LDAPObject.modify_ext_s(dn, modlist[, serverctrls=None[, clientctrls=None]]) → tuple

Performs an LDAP modify operation on an entry’s attributes. The dn argument is the distinguished name (DN) of the entry to modify, and modlist is a list of modifications to make to that entry.

Each element in the list modlist should be a tuple of the form (mod_op, mod_type, mod_vals), where mod_op indicates the operation (one of ldap.MOD_ADD, ldap.MOD_DELETE, or ldap.MOD_REPLACE), mod_type is a string indicating the attribute type name, and mod_vals is either a string value or a list of string values to add, delete or replace respectively. For the delete operation, mod_vals may be None indicating that all attributes are to be deleted.

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

The asynchronous methods modify() and modify_ext() return the message ID of the initiated request.

You might want to look into sub-module ldap.modlist for generating modlist.

The dn argument, and mod_type (second item) of modlist are text strings; see The bytes mode.

LDAPObject.modrdn(dn, newrdn[, delold=1]) → int
LDAPObject.modrdn_s(dn, newrdn[, delold=1]) → None

Perform a modify RDN operation, (i.e. a renaming operation). These routines take dn (the DN of the entry whose RDN is to be changed, and newrdn, the new RDN to give to the entry. The optional parameter delold is used to specify whether the old RDN should be kept as an attribute of the entry or not. The asynchronous version returns the initiated message id.

This operation is emulated by rename() and rename_s() methods since the modrdn2* routines in the C library are deprecated.
The `dn` and `newrdn` arguments are text strings; see *The bytes mode*.

```python
LDAPObject.passwd(user, oldpw, newpw[, serverctrls=None[, clientctrls=None ]]) → int
```

Perform a LDAP Password Modify Extended Operation operation on the entry specified by `user`. The old password in `oldpw` is replaced with the new password in `newpw` by a LDAP server supporting this operation.

If `oldpw` is not `None` it has to match the old password of the specified `user` which is sometimes used when a user changes his own password.

*serverctrls* and *clientctrls* like described in section *Arguments for LDAPv3 controls*.

The asynchronous version returns the initiated message id.

The `user`, `oldpw` and `newpw` arguments are text strings; see *The bytes mode*.

See also:

RFC 3062 - LDAP Password Modify Extended Operation

```python
LDAPObject.rename(dn, newrdn[, newsuperior=None[, delold=1[, serverctrls=None[, clientctrls=None ]]]]) → int
```

Perform a Rename operation, (i.e. a renaming operation). These routines take `dn` (the DN of the entry whose RDN is to be changed, and `newrdn`, the new RDN to give to the entry. The optional parameter `newsuperior` is used to specify a new parent DN for moving an entry in the tree (not all LDAP servers support this). The optional parameter `delold` is used to specify whether the old RDN should be kept as an attribute of the entry or not.

*serverctrls* and *clientctrls* like described in section *Arguments for LDAPv3 controls*.

The `dn` and `newdn` arguments are text strings; see *The bytes mode*.

```python
LDAPObject.result([msgid=RES_ANY[, all=1[, timeout=None ]]]) → 2-tuple
```

This method is used to wait for and return the result of an operation previously initiated by one of the LDAP asynchronous operations (e.g. `search()`, `modify()`, etc.)

The `msgid` parameter is the integer identifier returned by that method. The identifier is guaranteed to be unique across an LDAP session, and tells the `result()` method to request the result of that specific operation.

If a result is desired from any one of the in-progress operations, `msgid` should be specified as the constant `RES_ANY` and the method `result2()` should be used instead.

The `all` parameter only has meaning for `search()` responses and is used to select whether a single entry of the search response should be returned, or to wait for all the results of the search before returning.

A search response is made up of zero or more search entries followed by a search result. If `all` is 0, search entries will be returned one at a time as they come in, via separate calls to `result()`. If `all` is 1, the search response will be returned in its entirety, i.e. after all entries and the final search result have been received.

For `all` set to 0, result tuples trickle in (with the same message id), and with the result types `RES_SEARCH_ENTRY` and `RES_SEARCH_REFERENCE`, until the final result which has a result type of `RES_SEARCH_RESULT` and a (usually) empty data field. When `all` is set to 1, only one result is returned, with a result type of `RES_SEARCH_RESULT`, and all the result tuples listed in the data field.

The `timeout` parameter is a limit on the number of seconds that the method will wait for a response from the server. If `timeout` is negative (which is the default), the method will wait indefinitely for a response. The timeout can be expressed as a floating-point value, and a value of 0 effects a poll. If a timeout does occur, a `ldap.TIMEOUT` exception is raised, unless polling, in which case `(None, None)` is returned.
The `result()` method returns a tuple of the form (result-type, result-data). The first element, result-type is a string, being one of these module constants: RES_BIND, RES_SEARCH_ENTRY, RES_SEARCH_REFERENCE, RES_SEARCH_RESULT, RES_MODIFY, RES_ADD, RES_DELETE, RES_MODRDN, or RES_COMPARE.

If all is 0, one response at a time is returned on each call to `result()`, with termination indicated by result-data being an empty list.

See `search()` for a description of the search result’s result-data, otherwise the result-data is normally meaningless.

```python
LDAPObject.result2([msgid=RES_ANY, all=1, timeout=None]) → 3-tuple
```

This method behaves almost exactly like `result()`. But it returns a 3-tuple also containing the message id of the outstanding LDAP operation a particular result message belongs to. This is especially handy if one needs to dispatch results obtained with `msgid=RES_ANY` to several consumer threads which invoked a particular LDAP operation.

```python
LDAPObject.result3([msgid=RES_ANY, all=1, timeout=None]) → 4-tuple
```

This method behaves almost exactly like `result2()`. But it returns an extra item in the tuple, the decoded server controls.

```python
LDAPObject.result4([msgid=RES_ANY, all=1, timeout=None, add_ctrls=0, add_intermediates=0, add_extop=0, resp_ctrl_classes=None]) → 6-tuple
```

This method behaves almost exactly like `result3()`. But it returns an extra items in the tuple, the decoded results of an extended response.

The additional arguments are:
- `add_ctrls` (integer flag) specifies whether response controls are returned.
- `add_intermediates` (integer flag) specifies whether response controls of intermediate search results are returned.
- `add_extop` (integer flag) specifies whether the response of an extended operation is returned. If using extended operations you should consider using the method `extop_result()` or `extop_s()` instead.
- `resp_ctrl_classes` is a dictionary mapping the OID of a response controls to a `ldap.controls.ResponseControl` class of response controls known by the application. So the response control value will be automatically decoded. If None the global dictionary `ldap.controls.KNOWN_RESPONSE_CONTROLS` is used instead.

```python
LDAPObject.sasl_interactive_bind_s(who, auth[], serverctrls=None, clientctrls=None, sasl_flags=ldap.SASL_QUIET) → None
```

This call is used to bind to the directory with a SASL bind request. `auth` is an `ldap.sasl.sasl()` instance.

`serverctrls` and `clientctrls` like described in section Arguments for LDAPv3 controls.

```python
LDAPObject.sasl_non_interactive_bind_s(sasl_mech, serverctrls=None, clientctrls=None, sasl_flags=ldap.SASL_QUIET, authz_id="") → None
```

This call is used to bind to the directory with a SASL bind request with non-interactive SASL mechanism defined with argument `sasl_mech` and internally calls `sasl_interactive_bind_s()`.

`serverctrls` and `clientctrls` like described in section Arguments for LDAPv3 controls.

```python
LDAPObject.sasl_external_bind_s([serverctrls=None, clientctrls=None, sasl_flags=ldap.SASL_QUIET, authz_id=""]) → None
```

This call is used to bind to the directory with a SASL bind request with mechanism EXTERNAL and internally calls `sasl_non_interactive_bind_s()`.
serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

```
LDAPObject.sasl_gssapi_bind_s([serverctrls=None, clientctrls=None, sasl_flags=ldap.SASL_QUIET, authz_id=“”]) → None
```

This call is used to bind to the directory with a SASL bind request with mechanism GSSAPI and internally calls `sasl_non_interactive_bind_s()`.

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

```
LDAPObject.simple_bind([who=None, cred=None, serverctrls=None, clientctrls=None]) → int
```

After an LDAP object is created, and before any other operations can be attempted over the connection, a bind operation must be performed.

This method attempts to bind with the LDAP server using either simple authentication, or Kerberos (if available). The first and most general method, `bind()`, takes a third parameter, `method` which can currently solely be `AUTH_SIMPLE`.

```
LDAPObject.simple_bind_s([who=None, cred=None, serverctrls=None, clientctrls=None]) → None
```

changed in version 3.0: `simple_bind()` and `simple_bind_s()` now accept `None` for `who` and `cred`, too.

```
LDAPObject.search(base, scope[, filterstr=’(objectClass=*)‘][, attrlist=None[, attrsonly=0]]) → int
LDAPObject.search_s(base, scope[, filterstr=’(objectClass=*)‘][, attrlist=None[, attrsonly=0]]) → list|None
LDAPObject.search_st(base, scope[, filterstr=’(objectClass=*)‘][, attrlist=None[, attrsonly=0[, timeout=-1]]]) → list|None
LDAPObject.search_ext(base, scope[, filterstr=’(objectClass=*)‘][, attrlist=None[, attrsonly=0[, serverctrls=None[, clientctrls=None[, timeout=-1[, sizelimit=0]]]]]]][[...]]) → int
LDAPObject.search_ext_s(base, scope[, filterstr=’(objectClass=*)‘][, attrlist=None[, attrsonly=0[, serverctrls=None[, clientctrls=None[, timeout=-1[, sizelimit=0]]]]]]][[...]]) → list|None
```

Perform an LDAP search operation, with `base` as the DN of the entry at which to start the search, `scope` being one of `SCOPE_BASE` (to search the object itself), `SCOPE_ONELEVEL` (to search the object’s immediate children), or `SCOPE_SUBTREE` (to search the object and all its descendants).

The `filterstr` argument is a string representation of the filter to apply in the search.

See also:


Each result tuple is of the form `(dn, attrs)`, where `dn` is a string containing the DN (distinguished name) of the entry, and `attrs` is a dictionary containing the attributes associated with the entry. The keys of `attrs` are strings, and the associated values are lists of strings.

The DN in `dn` is automatically extracted using the underlying libldap function `ldap_get_dn()`, which may raise an exception if the DN is malformed.

If `attrsonly` is non-zero, the values of `attrs` will be meaningless (they are not transmitted in the result).

The retrieved attributes can be limited with the `attrlist` parameter. If `attrlist` is `None`, all the attributes of each entry are returned.
serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

The synchronous form with timeout, search_st() or search_ext_s(), will block for at most timeout seconds (or indefinitely if timeout is negative). A ldap.TIMEOUT exception is raised if no result is received within the specified time.

The amount of search results retrieved can be limited with the sizelimit parameter when using search_ext() or search_ext_s() (client-side search limit). If non-zero not more than sizelimit results are returned by the server.

The base and filterstr arguments, and attrlist contents, are text strings; see The bytes mode.

Changed in version 3.0: filterstr=None is equivalent to filterstr='(objectClass=*)'.

LDAPObject.start_tls_s() \rightarrow None

Negotiate TLS with server. The version attribute must have been set to VERSION3 (which it is by default) before calling this method. If TLS could not be started an exception will be raised.

See also:


LDAPObject.unbind() \rightarrow int

LDAPObject.unbind_s() \rightarrow None

LDAPObject.unbind_ext([serverctrls=None, clientctrls=None]) \rightarrow int

LDAPObject.unbind_ext_s([serverctrls=None, clientctrls=None]) \rightarrow None

This call is used to unbind from the directory, terminate the current association, and free resources. Once called, the connection to the LDAP server is closed and the LDAP object is marked invalid. Further invocation of methods on the object will yield exceptions.

serverctrls and clientctrls like described in section Arguments for LDAPv3 controls.

These methods are all synchronous in nature.

LDAPObject.whoami_s() \rightarrow string

This synchronous method implements the LDAP “Who Am I?” extended operation.

It is useful for finding out to find out which identity is assumed by the LDAP server after a SASL bind.

See also:


Connection-specific LDAP options

LDAPObject.get_option(option) \rightarrow int,string

This method returns the value of the LDAPObject option specified by option.

LDAPObject.set_option(option, invalue) \rightarrow None

This method sets the value of the LDAPObject option specified by option to invalue.

Object attributes

If the underlying library provides enough information, each LDAP object will also have the following attributes. These attributes are mutable unless described as read-only.
**LDAPObject.deref -> int**

Controls whether aliases are automatically dereferenced. This must be one of DEREF_NEVER, DEREF_SEARCHING, DEREF_FINDING or DEREF_ALWAYS. This option is mapped to option constant OPT_DEREF and used in the underlying OpenLDAP client lib.

**LDAPObject.network_timeout -> int**

Limit on waiting for a network response, in seconds. Defaults to NO_LIMIT. This option is mapped to option constant OPT_NETWORK_TIMEOUT and used in the underlying OpenLDAP client lib.

Changed in version 3.0.0: A timeout of -1 or None resets timeout to infinity.

**LDAPObject.protocol_version -> int**

Version of LDAP in use (either VERSION2 for LDAPv2 or VERSION3 for LDAPv3). This option is mapped to option constant OPT_PROTOCOL_VERSION and used in the underlying OpenLDAP client lib.

**LDAPObject.sizelimit -> int**

Limit on size of message to receive from server. Defaults to NO_LIMIT. This option is mapped to option constant OPT_SIZELIMIT and used in the underlying OpenLDAP client lib. Its use is deprecated in favour of sizelimit parameter when using search_ext().

**LDAPObject.timelimit -> int**

Limit on waiting for any response, in seconds. Defaults to NO_LIMIT. This option is mapped to option constant OPT_TIMELIMIT and used in the underlying OpenLDAP client lib. Its use is deprecated in favour of using timeout.

**LDAPObject.timeout -> int**

Limit on waiting for any response, in seconds. Defaults to NO_LIMIT. This option is used in the wrapper module.

---

**Example**

The following example demonstrates how to open a connection to an LDAP server using the ldap module and invoke a synchronous subtree search.

```python
>>> import ldap
>>> l = ldap.initialize('ldap://localhost:1390')
>>> l.search_s('ou=Testing,dc=stroeder,dc=de', ldap.SCOPE_SUBTREE, '(cn=fred*)', ['cn', 'mail'])
[('cn=Fred Feuerstein,ou=Testing,dc=stroeder,dc=de', {'cn': ['Fred Feuerstein']}],
>>> for dn, entry in r:
...     print('Processing', repr(dn))
...     handle_ldap_entry(entry)

5.3.2 ldap.asyncsearch Stream-processing of large search results

With newer Python versions one might want to consider using ldap.resiter instead.

Changed in version 3.0: In Python 3.7 async is a reserved keyword. The module ldap.async has been renamed to ldap.asyncsearch. The old name ldap.async is still available for backwards compatibility.
Deprecated since version 3.0: The old name `ldap.async` is deprecated, but will not be removed until Python 3.6 reaches end-of-life.

**Classes**

```python
class ldap.asyncsearch.AsyncSearchHandler(l)
    Class for stream-processing LDAP search results

    Arguments:
    l LDAPObject instance

    `afterFirstResult`()
        Do anything you want right after successfully receiving but before processing first result

    `postProcessing`()
        Do anything you want after receiving and processing all results

    `preProcessing`()
        Do anything you want after starting search but before receiving and processing results

    `processResults`(ignoreResultsNumber=0, processResultsCount=0, timeout=-1)
        `ignoreResultsNumber` Don’t process the first ignoreResultsNumber results.
        `processResultsCount` If non-zero this parameters indicates the number of results processed is limited to
        processResultsCount.
        `timeout` See parameter timeout of ldap.LDAPObject.result()

    `startSearch`(searchRoot, searchScope, filterStr, attrList=None, attrsOnly=0, timeout=-1, sizeLimit=0, serverctrls=None, clientctrls=None)
        `searchRoot` See parameter base of method LDAPObject.search()
        `searchScope` See parameter scope of method LDAPObject.search()
        `filterStr` See parameter filter of method LDAPObject.search()
        `attrList=None` See parameter attrlist of method LDAPObject.search()
        `attrsOnly` See parameter attrsonly of method LDAPObject.search()
        `timeout` Maximum time the server shall use for search operation
        `sizeLimit` Maximum number of entries a server should return (request client-side limit)
        `serverctrls` list of server-side LDAP controls
        `clientctrls` list of client-side LDAP controls
```

```python
class ldap.asyncsearch.List(l)
    Class for collecting all search results.

    This does not seem to make sense in the first place but think of retrieving exactly a certain portion of the available
    search results.
```

```python
class ldap.asyncsearch.Dict(l)
    Class for collecting all search results into a dictionary {dn:entry}
```

```python
class ldap.asyncsearch.IndexedDict(l, indexed_attrs=None)
    Class for collecting all search results into a dictionary {dn:entry} and maintain case-sensitive equality indexes
    to entries
```
class ldap.asyncsearch.LDIFWriter(l, writer_obj, headerStr=",", footerStr="")
Class for writing a stream LDAP search results to a LDIF file

Arguments:

l LDAPObject instance

writer_obj Either a file-like object or a ldif.LDIFWriter instance used for output

Examples

Using ldap.asyncsearch.List

This example demonstrates how to use class ldap.asyncsearch.List for retrieving partial search results even though the exception ldap.SIZELIMIT_EXCEEDED was raised because a server side limit was hit.

```python
import sys, ldap, ldap.asyncsearch

s = ldap.asyncsearch.List(
    ldap.initialize('ldap://localhost'),
)

s.startSearch(
    'dc=stroeder,dc=com',
    ldap.SCOPE_SUBTREE,
    '(objectClass=*)',
)

try:
    partial = s.processResults()
except ldap.SIZELIMIT_EXCEEDED:
    sys.stderr.write('Warning: Server-side size limit exceeded.
')
else:
    if partial:
        sys.stderr.write('Warning: Only partial results received.
')

sys.stdout.write(
    '%d results received.
' % (len(s.allResults))
)
```

Using ldap.asyncsearch.LDIFWriter

This example demonstrates how to use class ldap.asyncsearch.LDIFWriter for writing search results as LDIF to stdout.

```python
import sys, ldap, ldap.asyncsearch

s = ldap.asyncsearch.LDIFWriter(
    ldap.initialize('ldap://localhost:1390'),
    sys.stdout
)

s.startSearch(
    'dc=stroeder,dc=com',
    ldap.SCOPE_SUBTREE,
```
try:
    partial = s.processResults()
except ldap.SIZELIMIT_EXCEEDED:
    sys.stderr.write('Warning: Server-side size limit exceeded.\n')
else:
    if partial:
        sys.stderr.write('Warning: Only partial results received.\n')
sys.stderr.write('{d results received.\n' % (s.endResultBreak-s.beginResultsDropped
})

5.3.3 ldap.controls High-level access to LDAPv3 extended controls

Variables

ldap.controls.KNOWN_RESPONSE_CONTROLS
    Dictionary mapping the OIDs of known response controls to the accompanying ResponseControl classes. This is used by DecodeControlTuples() to automatically decode control values. Calling application can also register their custom ResponseControl classes in this dictionary possibly overriding pre-registered classes.

Classes

This module defines the following classes:

class ldap.controls.RequestControl (controlType=None, criticality=False, encodedControlValue=None)
    Base class for all request controls
    controlType OID as string of the LDAPv3 extended request control
criticality sets the criticality of the control (boolean)
encodedControlValue control value of the LDAPv3 extended request control (here it is the BER-encoded ASN.1 control value)
encodeControlValue()
    sets class attribute encodedControlValue to the BER-encoded ASN.1 control value composed by class attributes set before

class ldap.controls.ResponseControl (controlType=None, criticality=False)
    Base class for all response controls
    controlType OID as string of the LDAPv3 extended response control
criticality sets the criticality of the received control (boolean)
decodeControlValue(encodedControlValue)
    decodes the BER-encoded ASN.1 control value and sets the appropriate class attributes
class ldap.controls.LDAPControl (controlType=None, criticality=False, controlValue=None, encodedControlValue=None)

Base class for combined request/response controls mainly for backward-compatibility to python-ldap 2.3.x

Functions

This module defines the following functions:

ldap.controls.RequestControlTuples (ldapControls)

Return list of readily encoded 3-tuples which can be directly passed to C module _ldap

ldapControls sequence-type of RequestControl objects

ldap.controls.DecodeControlTuples (ldapControlTuples, knownLDAPControls=None)

Returns list of readily decoded ResponseControl objects

ldapControlTuples Sequence-type of 3-tuples returned by _ldap.result4() containing the encoded ASN.1 control values of response controls.

knownLDAPControls Dictionary mapping extended control’s OID to ResponseControl class of response controls known by the application. If None ldap.controls.KNOWN_RESPONSE_CONTROLS is used here.

Sub-modules

Various sub-modules implement specific LDAPv3 extended controls. The classes therein are derived from the base-classes ldap.controls.RequestControl, ldap.controls.ResponseControl or ldap.controls.LDAPControl.

Some of them require pyasn1 and pyasn1_modules to be installed:

Usually the names of the method arguments and the class attributes match the ASN.1 identifiers used in the specification. So looking at the referenced RFC or Internet-Draft is very helpful to understand the API.

ldap.controls.simple Very simple controls

class ldap.controls.simple.ValueLessRequestControl (controlType=None, criticality=False)

Base class for controls without a controlValue. The presence of the control in a LDAPv3 request changes the server’s behaviour when processing the request simply based on the controlType.

controlType OID of the request control

criticality criticality request control

class ldap.controls.simple-OctetStringInteger (controlType=None, criticality=False, integerValue=None)

Base class with controlValue being unsigend integer values

integerValue Integer to be sent as OctetString

class ldap.controls.simple.BooleanControl (controlType=None, criticality=False, booleanValue=False)

Base class for simple request controls with boolean control value.

Constructor argument and class attribute:

booleanValue Boolean (True/False or 1/0) which is the boolean controlValue.
```python
class ldap.controls.simple.ManageDSAITControl(criticality=False)
    Manage DSA IT Control

See also:

class ldap.controls.simple.RelaxRulesControl(criticality=False)
    Relax Rules Control

See also:
    draft-zeilenga-ldap-relax

class ldap.controls.simple.ProxyAuthzControl(criticality, authzId)
    Proxy Authorization Control

    authzId  string containing the authorization ID indicating the identity on behalf which the server should process
            the request

See also:

class ldap.controls.simple.AuthorizationIdentityRequestControl(criticality)
    Authorization Identity Request and Response Controls

See also:

class ldap.controls.simple.AuthorizationIdentityResponseControl(controlType=None, criticality=False)
    Authorization Identity Request and Response Controls

Class attributes:

    authzId  decoded authorization identity

See also:

class ldap.controls.simple.GetEffectiveRightsControl(criticality, authzId=None)
    Get Effective Rights Control

ldap.controls.libldap Various controls implemented in OpenLDAP libs

This module wraps C functions in OpenLDAP client libs which implement various request and response controls into
Python classes.

class ldap.controls.libldap.AssertionControl(criticality=True, filterstr='(objectClass=*)')
    LDAP Assertion control, as defined in RFC 4528

    filterstr  LDAP filter string specifying which assertions have to match so that the server processes the operation

See also:
    RFC 4528 - Lightweight Directory Access Protocol (LDAP) Assertion Control
```
class `ldap.controls.libldap.MatchValuesControl` *(criticality=False, \nstr=('objectClass=*'))* 

LDAP Matched Values control, as defined in RFC 3876 

`filterstr` LDAP filter string specifying which attribute values should be returned 

See also: 

RFC 3876 - Returning Matched Values with the Lightweight Directory Access Protocol version 3 (LDAPv3) 

class `ldap.controls.libldap.SimplePagedResultsControl` *(criticality=False, size=None, \ncookie=None)* 

LDAP Control Extension for Simple Paged Results Manipulation 

`size` Page size requested (number of entries to be returned) 

`cookie` Cookie string received with last page 

See also: 

RFC 2696 - LDAP Control Extension for Simple Paged Results Manipulation 

`ldap.controls.psearch` LDAP Persistent Search 

This module implements request and response controls for LDAP persistent search. 

See also: 

draft-ietf-ldapext-psearch 

`ldap.controls.sessiontrack` Session tracking control 

See also: 

draft-wahl-ldap-session 

`ldap.controls.readentry` Read entry control 

See also: 


5.3.4 `ldap.dn` LDAP Distinguished Name handling 

See also: 

For LDAPv3 DN syntax see: 


See also: 

For deprecated LDAPv2 DN syntax (obsoleted by LDAPv3) see: 

RFC 1779 - A String Representation of Distinguished Names 

The `ldap.dn` module defines the following functions:
ldap.dn.escape_dn_chars(s) → string
This function escapes characters in string s which are special in LDAP distinguished names. You should use this function when building LDAP DN strings from arbitrary input.

ldap.dn.str2dn(s[, flags=0 ]) → list
This function takes s and breaks it up into its component parts down to AVA level. The optional parameter flags describes the DN format of s (see DN format flags). Note that hex-encoded non-ASCII chars are decoded to the raw bytes.

Internally this function is implemented by calling OpenLDAP C function ldap_str2dn(3).

ldap.dn.str2dn(s[, flags=0 ]) → list
This function takes s and breaks it up into its component parts down to AVA level. The optional parameter flags describes the DN format of s (see DN format flags). Note that hex-encoded non-ASCII chars are decoded to the raw bytes.

Internally this function is implemented by calling OpenLDAP C function ldap_str2dn(3).

ldap.dn.dn2str(dn) → string
This function takes a decomposed DN in dn and returns a single string. It’s the inverse to str2dn(). Special characters are escaped with the help of function escape_dn_chars().

ldap.dn.explode_dn(dn[, notypes=False[, flags=0 ]]) → list
This function takes dn and breaks it up into its component parts. Each part is known as an RDN (Relative Distinguished Name). The optional notypes parameter is used to specify that only the RDN values be returned and not their types. The optional parameter flags describes the DN format of s (see DN format flags). This function is emulated by function str2dn() since the function ldap_explode_dn() in the C library is deprecated.

ldap.dn.explode_rdn(rdn[, notypes=False[, flags=0 ]]) → list
This function takes a (multi-valued) rdn and breaks it up into a list of characteristic attributes. The optional notypes parameter is used to specify that only the RDN values be returned and not their types. The optional flags parameter describes the DN format of s (see DN format flags). This function is emulated by function str2dn() since the function ldap_explode_rdn() in the C library is deprecated.

ldap.dn.is_dn(dn[, flags=0 ]) → boolean
This function checks whether dn is a valid LDAP distinguished name by passing it to function str2dn().

Examples
Splitting a LDAPv3 DN to AVA level. Note that both examples have the same result but in the first example the non-ASCII chars are passed as is (byte buffer string) whereas in the second example the hex-encoded DN representation are passed to the function.

```
>>> ldap.dn.str2dn('cn=Michael Str\xc3\xb6der,dc=example,dc=com',flags=ldap.DN_FORMAT_˓→LDAPV3)
[['cn', 'Michael Str\xc3\xb6der', 4]],[['dc', 'example', 1]],[['dc', 'com', 1]]
>>> ldap.dn.str2dn('cn=Michael Str\C3\B6der,dc=example,dc=com',flags=ldap.DN_FORMAT_˓→LDAPV3)
[['cn', 'Michael Str\xc3\xb6der', 4]],[['dc', 'example', 1]],[['dc', 'com', 1]]
```

Splitting a LDAPv2 DN into RDN parts:
```
>>> ldap.dn.explode_dn('cn=John Doe;dc=example;dc=com',flags=ldap.DN_FORMAT_LDAPV2)
['cn=John Doe', 'dc=example', 'dc=com']
```

Splitting a multi-valued RDN:
```
>>> ldap.dn.explode_rdn('cn=John Doe+mail=john.doe@example.com',flags=ldap.DN_FORMAT_˓→LDAPV2)
['cn=John Doe', 'mail=john.doe@example.com']
```

Splitting a LDAPv3 DN with a multi-valued RDN into its AVA parts:
```
>>> ldap.dn.str2dn('cn=John Doe+mail=john.doe@example.com,dc=example,dc=com')
[['cn', 'John Doe', 1], ['mail', 'john.doe@example.com', 1]],[['dc', 'example', 1]],[['dc', 'com', 1]]
```
5.3.5 ldap.extop High-level access to LDAPv3 extended operations

Classes

This module defines the following classes:

```python
class ldap.extop.ExtendedRequest(requestName, requestValue)
    Generic base class for a LDAPv3 extended operation request
    
    requestName  OID as string of the LDAPv3 extended operation request
    requestValue value of the LDAPv3 extended operation request (here it is the BER-encoded ASN.1 request value)
    
    encodedRequestValue()
    returns the BER-encoded ASN.1 request value composed by class attributes set before
```

```python
class ldap.extop.ExtendedResponse(responseName, encodedResponseValue)
    Generic base class for a LDAPv3 extended operation response
    
    responseName  OID as string of the LDAPv3 extended operation response
    encodedResponseValue  BER-encoded ASN.1 value of the LDAPv3 extended operation response
    
    decodeResponseValue(value)
    decodes the BER-encoded ASN.1 extended operation response value and sets the appropriate class attributes
```

```python
ldap.extop.dds Classes for Dynamic Entries extended operations
```

This requires `pyasn1` and `pyasn1_modules` to be installed.

See also:


5.3.6 ldap.filter LDAP filter handling

See also:


The `ldap.filter` module defines the following functions:

```python
ldap.filter.escape_filter_chars(assertion_value[, escape_mode=0])
    This function escapes characters in `assertion_value` which are special in LDAP filters. You should use this function when building LDAP filter strings from arbitrary input. `escape_mode` means: If 0 only special chars mentioned in RFC 4515 are escaped. If 1 all NON-ASCII chars are escaped. If 2 all chars are escaped.
```

```python
ldap.filter.filter_format(filter_template, assertion_values)
    This function applies `escape_filter_chars()` to each of the strings in list `assertion_values`. After that `filter_template` containing as many %s placeholders as count of assertion values is used to build the whole filter string.
```
5.3.7 ldap.modlist Generate modify lists

The `ldap.modlist` module defines the following functions:

- `ldap.modlist.addModlist(entry[, ignore_attr_types=[]]) → list`
  
  This function builds a list suitable for passing it directly as argument `modlist` to method `ldap.ldapobject.LDAPObject.add()` or its synchronous counterpart `ldap.ldapobject.LDAPObject.add_s()`.

  - `entry` is a dictionary like returned when receiving search results.
  - `ignore_attr_types` is a list of attribute type names which shall be ignored completely. Attributes of these types will not appear in the result at all.

- `ldap.modlist.modifyModlist(olde_entry, new_entry[, ignore_attr_types=[], ignore_oldexistent=0[, case_ignore_attr_types=None]]) → list`
  
  This function builds a list suitable for passing it directly as argument `modlist` to method `ldap.ldapobject.LDAPObjectmodify()` or its synchronous counterpart `ldap.ldapobject.LDAPObject.modify_s()`.

  Roughly when applying the resulting modify list to an entry holding the data `olde_entry` it will be modified in such a way that the entry holds `new_entry` after the modify operation. It is handy in situations when it is impossible to track user changes to an entry’s data or for synchronizing operations.

  - `olde_entry` and `new_entry` are dictionaries like returned when receiving search results.
  - `ignore_attr_types` is a list of attribute type names which shall be ignored completely. These attribute types will not appear in the result at all.
  - `ignore_oldexistent` is non-zero attribute type names which are in `olde_entry` but are not found in `new_entry` at all are not deleted. This is handy for situations where your application sets attribute value to an empty string for deleting an attribute. In most cases leave zero.
  - `case_ignore_attr_types` is a list of attribute type names for which the comparison will be conducted case-insensitive. It is useful in situations where a LDAP server normalizes values and one wants to avoid unnecessary changes (e.g. case of attribute type names in DNs).

  **Note:** Replacing attribute values is always done with a `ldap.MOD_DELETE/ldap.MOD_ADD` pair instead of `ldap.MOD_REPLACE` to work-around potential issues with attributes for which no EQUALITY matching rule are defined in the server's subschema. This works correctly in most situations but rarely fails with some LDAP servers implementing (schema) checks on transient state entry during processing the modify operation.

5.3.8 ldap.resiter Generator for stream-processing of large search results

**class ldap.resiter.ResultProcessor**

This is a mix-in class to be used with class `ldap.LDAPObject` or derived classes which has these methods:

- `ResultProcessor.allresults(msgid, timeout=-1, add_ctrls=0)`
  
  Generator function which returns an iterator for processing all LDAP operation results of the given `msgid` like retrieved with `LDAPObject.result3()` -> 4-tuple
Examples

Using ldap.resiter.ResultProcessor

This example demonstrates how to use mix-in class ldap.resiter.ResultProcessor for retrieving results formerly requested with `ldap.LDAPObject.search()` and processing them in a for-loop.

```python
import sys, ldap, ldap.resiter

class MyLDAPObject(ldap.ldapobject.LDAPObject, ldap.resiter.ResultProcessor):
    pass

l = MyLDAPObject('ldap://localhost')

# Asynchronous search method
msg_id = l.search('dc=stroeder,dc=com', ldap.SCOPE_SUBTREE, '(objectClass=*)')

for res_type, res_data, res_msgid, res_controls in l.allresults(msg_id):
    for dn, entry in res_data:
        # process dn and entry
        print(dn, entry['objectClass'])
```

5.3.9 `ldap.schema` Handling LDAPv3 schema

This module deals with schema information usually retrieved from a special subschema subentry provided by the server. It is closely modeled along the directory information model described in the following RFC with which you should make yourself familiar when trying to use this module:

See also:


`ldap.schema.subentry` Processing LDAPv3 subschema subentry

`ldap.schema.subentry.NOT_HUMAN_READABLE_LDAP_SYNTAXES`

Dictionary where the keys are the OIDs of LDAP syntaxes known to be not human-readable when displayed to a console without conversion and which cannot be decoded to a `types.UnicodeType`.

Functions

`ldap.schema.subentry.urlfetch(uri, trace_level=0)`

Fetches a parsed schema entry by uri.

If uri is a LDAP URL the LDAP server is queried directly. Otherwise uri is assumed to point to a LDIF file which is loaded with urllib.

Classes

`class ldap.schema.subentry.SubSchema(sub_schema_sub_entry, check_uniqueness=1)`

Arguments:

`sub_schema_sub_entry` Dictionary usually returned by LDAP search or the LDIF parser containing the sub schema sub entry
check_uniqueness  Defines whether uniqueness of OIDs and NAME is checked.

0  no check
1  check but add schema description with work-around
2  check and raise exception if non-unique OID or NAME is found

Class attributes:
sed  Dictionary holding the subschema information as pre-parsed SchemaElement objects (do not access directly!)
name2oid  Dictionary holding the mapping from NAMEs to OIDs (do not access directly!)
non_unique_oids  List of OIDs used at least twice in the subschema
non_unique_names  List of NAMEs used at least twice in the subschema for the same schema element

attribute_types (object_class_list, attr_type_filter=None, raise_keyerror=1, ignore_dit_content_rule=0)
Returns a 2-tuple of all must and may attributes including all inherited attributes of superior object classes by walking up classes along the SUP attribute.
The attributes are stored in a ldap.cidict.cidict dictionary.

object_class_list  list of strings specifying object class names or OIDs
attr_type_filter  list of 2-tuples containing lists of class attributes which has to be matched
raise_keyerror  All KeyError exceptions for non-existent schema elements are ignored
ignore_dit_content_rule  A DIT content rule governing the structural object class is ignored

get_applicable_aux_classes (nameoroid)
Return a list of the applicable AUXILIARY object classes for a STRUCTURAL object class specified by 'nameoroid' if the object class is governed by a DIT content rule. If there's no DIT content rule all available AUXILIARY object classes are returned.

get_inheritedattr (se_class, nameoroid, name)
Get a possibly inherited attribute specified by name of a schema element specified by nameoroid. Returns None if class attribute is not set at all.
Raises KeyError if no schema element is found by nameoroid.

get_inheritedobj (se_class, nameoroid, inherited=None)
Get a schema element by name or OID with all class attributes set including inherited class attributes

get_obj (se_class, nameoroid, default=None, raise_keyerror=0)
Get a schema element by name or OID

get_structural_oc (oc_list)
Returns OID of structural object class in oc_list if any is present. Returns None else.

get_syntax (nameoroid)
Get the syntax of an attribute type specified by name or OID

getooid (se_class, nameoroid, raise_keyerror=0)
Get an OID by name or OID

ldap_entry ()
Returns a dictionary containing the sub schema sub entry

listall (schema_element_class, schema_element_filters=None)
Returns a list of OIDs of all available schema elements of a given schema element class.
**tree** *(schema_element_class, schema_element_filters=None)*

Returns a ldap.cidict.cidict dictionary representing the tree structure of the schema elements.

**ldap.schema.models Schema elements**

**class** *ldap.schema.models.Entry (schema, dn, entry)*

Schema-aware implementation of an LDAP entry class.

Mainly it holds the attributes in a string-keyed dictionary with the OID as key.

**attribute_types** *(attr_type_filter=None, raise_keyerror=1)*

Convenience wrapper around SubSchema.attribute_types() which passes object classes of this particular entry as argument to SubSchema.attribute_types()

**class** *ldap.schema.models.SchemaElement (schema_element_str=None)*

Base class for all schema element classes. Not used directly!

Arguments:

**schema_element_str** String which contains the schema element description to be parsed. (Bytestrings are decoded using UTF-8)

Class attributes:

**schema_attribute** LDAP attribute type containing a certain schema element description

**token_defaults** Dictionary internally used by the schema element parser containing the defaults for certain schema description key-words

**class** *ldap.schema.models.AttributeType (schema_element_str=None)*

Arguments:

**schema_element_str** String containing an AttributeTypeDescription

Class attributes:

**oid** OID assigned to the attribute type

**names** This list of strings contains all NAMEs of the attribute type

**desc** This string contains description text (DESC) of the attribute type

**obsolete** Integer flag (0 or 1) indicating whether the attribute type is marked as OBSOLETE in the schema

**single_value** Integer flag (0 or 1) indicating whether the attribute must have only one value

**syntax** String contains OID of the LDAP syntax assigned to the attribute type

**no_user_mod** Integer flag (0 or 1) indicating whether the attribute is modifiable by a client application

**equality** String contains NAME or OID of the matching rule used for checking whether attribute values are equal

**substr** String contains NAME or OID of the matching rule used for checking whether an attribute value contains another value

**ordering** String contains NAME or OID of the matching rule used for checking whether attribute values are lesser-equal than

**usage** USAGE of an attribute type: 0 = userApplications, 1 = directoryOperation, 2 = distributedOperation, 3 = dSAOperation

**sup** This list of strings contains NAMEs or OIDs of attribute types this attribute type is derived from
**class** `ldap.schema.models.ObjectClass` *(schema_element_str=None)*
Arguments:

- `schema_element_str` String containing an ObjectClassDescription
Class attributes:

- `oid` OID assigned to the object class
- `names` This list of strings contains all NAMEs of the object class
- `desc` This string contains description text (DESC) of the object class
- `obsolete` Integer flag (0 or 1) indicating whether the object class is marked as OBSOLETE in the schema
- `must` This list of strings contains NAMEs or OIDs of all attributes an entry of the object class must have
- `may` This list of strings contains NAMEs or OIDs of additional attributes an entry of the object class may have
- `kind` Kind of an object class: 0 = STRUCTURAL, 1 = ABSTRACT, 2 = AUXILIARY
- `sup` This list of strings contains NAMEs or OIDs of object classes this object class is derived from

**class** `ldap.schema.models.MatchingRule` *(schema_element_str=None)*
Arguments:

- `schema_element_str` String containing an MatchingRuleDescription
Class attributes:

- `oid` OID assigned to the matching rule
- `names` This list of strings contains all NAMEs of the matching rule
- `desc` This string contains description text (DESC) of the matching rule
- `obsolete` Integer flag (0 or 1) indicating whether the matching rule is marked as OBSOLETE in the schema
- `syntax` String contains OID of the LDAP syntax this matching rule is usable with

**class** `ldap.schema.models.MatchingRuleUse` *(schema_element_str=None)*
Arguments:

- `schema_element_str` String containing an MatchingRuleUseDescription
Class attributes:

- `oid` OID of the accompanying matching rule
- `names` This list of strings contains all NAMEs of the matching rule
- `desc` This string contains description text (DESC) of the matching rule
- `obsolete` Integer flag (0 or 1) indicating whether the matching rule is marked as OBSOLETE in the schema
- `applies` This list of strings contains NAMEs or OIDs of attribute types for which this matching rule is used

**class** `ldap.schema.models.DITContentRule` *(schema_element_str=None)*
Arguments:

- `schema_element_str` String containing an DITContentRuleDescription
Class attributes:

- `oid` OID of the accompanying structural object class
- `names` This list of strings contains all NAMEs of the DIT content rule
- `desc` This string contains description text (DESC) of the DIT content rule
obsolete  Integer flag (0 or 1) indicating whether the DIT content rule is marked as OBSOLETE in the schema

aux  This list of strings contains NAMES or OIDs of all auxiliary object classes usable in an entry of the object class

must  This list of strings contains NAMES or OIDs of all attributes an entry of the object class must have which may extend the list of required attributes of the object classes of an entry

may  This list of strings contains NAMES or OIDs of additional attributes an entry of the object class may have which may extend the list of optional attributes of the object classes of an entry

nots  This list of strings contains NAMES or OIDs of attributes which may not be present in an entry of the object class

class  ldap.schema.models.NameForm(schema_element_str=None)

Arguments:

    schema_element_str  String containing an NameFormDescription

Class attributes:

    oid  OID of the name form

    names  This list of strings contains all NAMES of the name form

    desc  This string contains description text (DESC) of the name form

    obsolete  Integer flag (0 or 1) indicating whether the name form is marked as OBSOLETE in the schema

    form  List of strings with NAMES or OIDs of associated name forms

    oc  String with NAME or OID of structural object classes this name form is usable with

    must  This list of strings contains NAMES or OIDs of all attributes an RDN must contain

    may  This list of strings contains NAMES or OIDs of additional attributes an RDN may contain

class  ldap.schema.models.DITStructureRule(schema_element_str=None)

Arguments:

    schema_element_str  String containing an DITStructureRuleDescription

Class attributes:

    ruleid  rule ID of the DIT structure rule (only locally unique)

    names  This list of strings contains all NAMES of the DIT structure rule

    desc  This string contains description text (DESC) of the DIT structure rule

    obsolete  Integer flag (0 or 1) indicating whether the DIT content rule is marked as OBSOLETE in the schema

    form  List of strings with NAMES or OIDs of associated name forms

    sup  List of strings with NAMES or OIDs of allowed structural object classes of superior entries in the DIT

Examples for ldap.schema

import  ldap.schema
5.3.10 `ldap.syncrepl` Implementation of a syncrepl consumer

See also:


This requires `pyasn1` and `pyasn1_modules` to be installed.

Classes

This module defines the following classes:

5.3.11 `ldap.sasl` SASL Authentication Methods

This module implements various authentication methods for SASL bind.

See also:

RFC 4422 - Simple Authentication and Security Layer (SASL)

Constants

```plaintext
ldap.sasl.CB_USER
ldap.sasl.CB_AUTHNAME
ldap.sasl.CB_LANGUAGE
ldap.sasl.CB_PASS
ldap.sasl.CB_ECHOPROMPT
ldap.sasl.CB_NOECHOPROMPT
ldap.sasl.CB_GETREALM
```

Classes

```python
class ldap.sasl.sasl(cb_value_dict, mech):
    This class handles SASL interactions for authentication. If an instance of this class is passed to Ldap's
    sasl_bind_s() method, the library will call its callback() method. For specific SASL authentication mechanisms,
    this method can be overridden
    This class is used with `ldap.LDAPObject.sasl_interactive_bind_s()`.

callback(cb_id, challenge, prompt, defresult)
    The callback method will be called by the sasl_bind_s() method several times. Each time it will provide
    the id, which tells us what kind of information is requested (the CB_* constants above). The challenge
    might be a short (English) text or some binary string, from which the return value is calculated. The prompt
    argument is always a human-readable description string; The defresult is a default value provided by the
    sasl library
    Currently, we do not use the challenge and prompt information, and return only information which is stored
    in the self.cb_value_dict cb_value_dictionary. Note that the current callback interface is not very useful
    for writing generic sasl GUIs, which would need to know all the questions to ask, before the answers are
    returned to the sasl lib (in contrast to one question at a time).
```
Unicode strings are always converted to bytes.

class ldap.sasl.cram_md5 (authc_id, password, authz_id="")
This class handles SASL CRAM-MD5 authentication.

class ldap.sasl.digest_md5 (authc_id, password, authz_id="")
This class handles SASL DIGEST-MD5 authentication.

class ldap.sasl.gssapi (authz_id="")
This class handles SASL GSSAPI (i.e. Kerberos V) authentication.
You might consider using convenience method ldap.LDAPObject.sasl_gssapi_bind_s().

class ldap.sasl.external (authz_id="")
This class handles SASL EXTERNAL authentication (i.e. X.509 client certificate)
You might consider using convenience method ldap.LDAPObject.sasl_external_bind_s().

Examples for ldap.sasl

This example connects to an OpenLDAP server via LDAP over IPC (see draft-chu-ldap-ldapi) and sends a SASL external bind request.

```python
import ldap, ldap.sasl, urllib

ldapi_path = '/tmp/openldap-socket'
ldap_conn = ldap.initialize('ldapi://%s' % (urllib.quote_plus(ldapi_path))
# Send SASL bind request for mechanism EXTERNAL
ldap_conn.sasl_non_interactive_bind_s('EXTERNAL')
# Find out the SASL Authorization Identity
print ldap_conn.whoami_s()
```

5.3.12 ldif LDIF parser and generator

This module parses and generates LDAP data in the format LDIF. It is implemented in pure Python and does not rely on any non-standard modules. Therefore it can be used stand-alone without the rest of the python-ldap package.

See also:

RFC 2849 - The LDAP Data Interchange Format (LDIF) - Technical Specification

Functions

ldif.CreateLDIF (dn, record, base64_attr=None, cols=76)
Create LDIF single formatted record including trailing empty line. This is a compatibility function.

dn  string-representation of distinguished name
record Either a dictionary holding the LDAP entry {attrtype:record} or a list with a modify list like for
LDAPObjec MODIFY().
base64_attr list of attribute types to be base64-encoded in any case
cols Specifies how many columns a line may have before it’s folded into many lines.
Deprecated since version 3.0: `ldif.CreateLDIF()` is deprecated. It will be removed in version 3.1. Use `ldif.LDIFWriter.unparse()` with a file or `io.StringIO` instead.

```python
ldif.ParseLDIF(f, ignore_attrs=None, maxentries=0)
```
Parse LDIF records read from file. This is a compatibility function.

Deprecated since version 3.0: `ldif.ParseLDIF()` is deprecated. It will be removed in version 3.1. Use the `all_records` attribute of the returned value of `ldif.LDIFRecordList.parse()` instead.

## Classes

**class ldif.LDIFWriter**(output_file, base64_attrs=None, cols=76, line_sep='n')

Write LDIF entry or change records to file object Copy LDIF input to a file output object containing all data retrieved via URLs

```python
unparse(dn, record)
```

- **dn** string-representation of distinguished name
- **record** Either a dictionary holding the LDAP entry `attrtype:record` or a list with a modify list like for LDAPObject.modify().

**class ldif.LDIFParser**(input_file, ignored_attr_types=None, max_entries=0, process_url_schemes=None, line_sep='n')

Base class for a LDIF parser. Applications should sub-class this class and override method handle() to implement something meaningful.

Public class attributes:

- **records_read** Counter for records processed so far

```python
handle(dn, entry)
```

Process a single content LDIF record. This method should be implemented by applications using LDIFParser.

```python
handle_modify(dn, modops, controls=None)
```

Process a single LDIF record representing a single modify operation. This method should be implemented by applications using LDIFParser.

```python
parse()
```

Invokes LDIFParser.parse_entry_records() for backward compatibility

```python
parse_entry_records()
```
Continuously read and parse LDIF entry records

**class ldif.LDIFRecordList**(input_file, ignored_attr_types=None, max_entries=0, process_url_schemes=None)

Collect all records of a LDIF file. It can be a memory hog!

Records are stored in `all_records` as a single list of 2-tuples (dn, entry), after calling `parse()`.

- **all_records = None**
  List storing parsed records.

```python
handle(dn, entry)
```

Append a single record to the list of all records (`all_records`).

```python
handle_modify(dn, modops, controls=None)
```

Process a single LDIF record representing a single modify operation. This method should be implemented by applications using LDIFParser.
class `ldif.LDIFCopy` *(input_file, output_file, ignored_attr_types=None, max_entries=0, process_url_schemes=None, base64_attrs=None, cols=76, line_sep='n')*

Copy LDIF input to LDIF output containing all data retrieved via URLs

**`handle(dn, entry)`**

Write single LDIF record to output file.

**Example**

The following example demonstrates how to write LDIF output of an LDAP entry with `ldif` module.

```python
>>> import sys
>>> entry={'objectClass':['top','person'],'cn':['Michael Stroeder'],'sn':['Stroeder']}
>>> dn='cn=Michael Stroeder,ou=Test'
>>> ldif_writer=ldif.LDIFWriter(sys.stdout)
>>> ldif_writer.unparse(dn,entry)
dn: cn=Michael Stroeder,ou=Test
cn: Michael Stroeder
objectClass: top
objectClass: person
sn: Stroeder
```

The following example demonstrates how to parse an LDIF file with `ldif` module, skip some entries and write the result to stdout.

```python
import sys
from ldif import LDIFParser,LDIFWriter

SKIP_DN = ["uid=foo,ou=People,dc=example,dc=com",
        "uid=bar,ou=People,dc=example,dc=com"]

class MyLDIF(LDIFParser):
    def __init__(self,input,output):
        LDIFParser.__init__(self,input)
        self.writer = LDIFWriter(output)

    def handle(self,dn,entry):
        if dn in SKIP_DN:
            return
        self.writer.unparse(dn,entry)

parser = MyLDIF(open("input.ldif", 'rb'), sys.stdout)
parser.parse()
```

**5.3.13 ldapurl LDAP URL handling**

This module parses and generates LDAP URLs. It is implemented in pure Python and does not rely on any non-standard modules. Therefore it can be used stand-alone without the rest of the python-ldap package. Compatibility note: This module has been solely tested on Python 2.x and above.

See also:

RFC 4516 - The LDAP URL Format
Constants

The `ldapurl` module exports the following constants:

**ldapurl.SEARCH_SCOPE**
This dictionary maps a search scope string identifier to the corresponding integer value used with search operations in `ldap`.

**ldapurl.SEARCH_SCOPE_STR**
This dictionary is the inverse to `SEARCH_SCOPE`. It maps a search scope integer value to the corresponding string identifier used in a LDAP URL string representation.

**ldapurl.LDAP_SCOPE_BASE**

**ldapurl.LDAP_SCOPE_ONELEVEL**

**ldapurl.LDAP_SCOPE_SUBTREE**

Functions

**ldapurl.isLDAPUrl(s)**
Returns 1 if s is a LDAP URL, 0 else

**ldapurl.ldapUrlEscape(s)**
Returns URL encoding of string s

Classes

LDAP URLs

A `LDAPUrl` object represents a complete LDAP URL.

**class ldapurl.LDAPUrl(ladapUrl=None, urlscheme='ldap', hostport='', dn='', attrs=None, scope=None, filterstr=None, extensions=None, who=None, cred=None)**

Class for parsing and unparsing LDAP URLs as described in RFC 4516.

Usable class attributes:

- `urlscheme` URL scheme (either ldap, ldaps or ldapi)
- `hostport` LDAP host (default '')
- `dn` String holding distinguished name (default '')
- `attrs` list of attribute types (default None)
- `scope` integer search scope for ldap-module
- `filterstr` String representation of LDAP Search Filters (see RFC 4515)
- `extensions` Dictionary used as extensions store
- `who` Maps automagically to bindname LDAP URL extension
- `cred` Maps automagically to X-BINDPW LDAP URL extension

**applyDefaults(defaults)**
Apply defaults to all class attributes which are None.

**defaults** Dictionary containing a mapping from class attributes to default values
htmlHREF (urlPrefix=None, hrefText=None, hrefTarget=None)

Returns a string with HTML link for this LDAP URL.

- urlPrefix  Prefix before LDAP URL (e.g. for addressing another web-based client)
- hrefText   link text/description
- hrefTarget string added as link target attribute

initializeUrl()

Returns LDAP URL suitable to be passed to ldap.initialize()

unparse()

Returns LDAP URL depending on class attributes set.

LDAP URL extensions

A LDAPUrlExtension object represents a single LDAP URL extension whereas LDAPUrlExtensions represents a list of LDAP URL extensions.

class ldapurl.LDAPUrlExtension (extensionStr=None, critical=0, extype=None, exvalue=None)

Class for parsing and unparsing LDAP URL extensions as described in RFC 4516.

Usable class attributes:

- critical  Boolean integer marking the extension as critical
- extype    Type of extension
- exvalue   Value of extension

class ldapurl.LDAPUrlExtensions (default=None)

Models a collection of LDAP URL extensions as dictionary type

Example

Important security advice: For security reasons you should not specify passwords in LDAP URLs unless you really know what you are doing.

The following example demonstrates how to parse a LDAP URL with ldapurl module.

```python
>>> import ldapurl
>>> ldap_url = ldapurl.LDAPUrl('ldap://localhost:1389/dc=stroeder,dc=com?cn,mail???
˓→bindname=cn=Michael%2c,dc=stroeder%2c,dc=com,X-BINDPW=secret')
>>> # Using the parsed LDAP URL by reading the class attributes
>>> ldap_url.dn
'dc=stroeder,dc=com'
>>> ldap_url.hostport
'localhost:1389'
>>> ldap_url.attrs
['cn','mail']
>>> ldap_url.filterstr
'(objectclass=*)'
>>> ldap_url.who
'cn=Michael,dc=stroeder,dc=com'
>>> ldap_url.cred
'secret'
>>> ldap_url.scope
0
```
The following example demonstrates how to generate a LDAP URL with module `ldapurl` module.

```python
>>> import ldapurl
>>> ldap_url = ldapurl.LDAPUrl(hostport='localhost:1389', dn='dc=stroeder,dc=com', attrs=['cn','mail'], who='cn=Michael,dc=stroeder,dc=com', cred='secret')
>>> ldap_url.unparse()
'ldap://localhost:1389/dc=stroeder,dc=com?cn,mail?base?(objectclass=*)?bindname=cn=Michael%2Cdc=stroeder%2Cdc=com,X-BINDPW=secret'
```

### 5.3.14 `slapdtest` Spawning test instances of OpenLDAP’s slapd server

The module is used for testing python-ldap itself but can be used for automatically testing any OpenLDAP-based configuration setup.

This module is pure Python and does not rely on any non-standard modules. Therefore it can be used stand-alone without the rest of the python-ldap package.

**Functions**

**Classes**

```python
class slapdtest.SlapdObject
    Controller class for a slapd instance, OpenLDAP’s server.
    This class creates a temporary data store for slapd, runs it listening on a private Unix domain socket and TCP port, and initializes it with a top-level entry and the root user.
    When a reference to an instance of this class is lost, the slapd server is shut down.
    ```
```python
gen_config()
    generates a slapd.conf and returns it as one string
    for generating specific static configuration files you have to override this method

ldapadd(ldif, extra_args=None)
    Runs ldapadd on this slapd instance, passing it the ldif content

ldapdelete(dn, recursive=False, extra_args=None)
    Runs ldapdelete on this slapd instance, deleting ‘dn’

ldapmodify(ldif, extra_args=None)
    Runs ldapmodify on this slapd instance, passing it the ldif content

ldapwhoami(extra_args=None)
    Runs ldapwhoami on this slapd instance

restart()
    Restarts the slapd server with same data

setup_rundir()
    creates rundir structure
    for setting up a custom directory structure you have to override this method

start()
    Starts the slapd server process running, and waits for it to come up.

stop()
    Stops the slapd server, and waits for it to terminate and cleans up
```
wait()
    Waits for the slapd process to terminate by itself.

class slapdtest.SlapdTestCase(methodName='runTest')
    test class which also clones or initializes a running slapd

server_class
    alias of SlapdObject

5.4 Third-party documentation

The following documents referenced are not written by python-ldap project members. Therefore some information might be outdated or links might be broken.

5.4.1 Python LDAP Applications articles by Matt Butcher

- Part 1 - Installing and Configuring the Python-LDAP Library and Binding to an LDAP Directory
  This also covers SASL.
- Part 2 - LDAP Operations
- Part 3 - More LDAP Operations and the LDAP URL Library
- Part 4 - LDAP Schema
  Gee, someone waded through the badly documented mysteries of module ldap.schema.

5.4.2 LDAP Programming in Python

Another article for getting started with python-ldap.

5.4.3 RFC 1823

The LDAP Application Program Interface, mainly for LDAPv2.

5.4.4 LDAPEXT draft

The Internet draft of the discontinued IETF working group LDAPEXT is of interest here since the OpenLDAP 2 libs implement this (expired) draft.

5.4.5 OpenLDAP

It’s worth to have a look at the manual pages and the Developer’s FAQ.

5.4.6 VSLDAP

VSLDAP Interoperability Test Suite.
5.5 Contributing to python-ldap

Thank you for your interest in python-ldap! If you’d like to contribute (be it code, documentation, maintenance effort, or anything else), this guide is for you.

5.5.1 Sample workflow for python-ldap development

This document will guide you through the process of contributing a change to python-ldap.

We assume that, as a user of python-ldap, you’re not new to software development in general, so these instructions are terse. If you need additional detail, please do ask on the mailing list.

**Note:** The following instructions are for Linux. If you can translate them to another system, please contribute your translation!

Install Git, tox and the **Build prerequisites**.

Clone the repository:

```bash
$ git clone https://github.com/python-ldap/python-ldap
$ cd python-ldap
```

Create a virtual environment to ensure you in-development python-ldap won’t affect the rest of your system:

```bash
$ python3 -m venv __venv__
```

(For Python 2, install virtualenv and use it instead of `python3 -m venv`.)

Activate the virtual environment:

```bash
$ source __venv__/bin/activate
```

Install python-ldap to it in editable mode:

```bash
(__venv__)$ python -m pip install -e .
```

This way, importing a Python module from python-ldap will directly use the code from your source tree. If you change C code, you will still need to recompile (using the `pip install` command again).

Change the code as desired.

To run tests, install and run tox:

```bash
(__venv__)$ python -m pip install tox
(__venv__)$ tox --skip-missing-interpreters
```

This will run tests on all supported versions of Python that you have installed, skipping the ones you don’t. To run a subset of test environments, run for example:

```bash
(__venv__)$ tox -e py27,py36
```

In addition to `pyXY` environments, we have extra environments for checking things independent of the Python version:

- `doc` checks syntax and spelling of the documentation
- `coverage-report` generates a test coverage report for Python code. It must be used last, e.g. `tox -e py27,py36,coverage-report`. 

5.5. Contributing to python-ldap
• py2-nosasltls and py3-nosasltls check functionality without SASL and TLS bindings compiled in.

When your change is ready, commit to Git, and submit a pull request on GitHub. You can take a look at the Instructions for core committers to see what we are looking for in a pull request.

If you don’t want to open a GitHub account, please send patches as attachments to the python-ldap mailing list.

5.5.2 Communication

Always keep in mind that python-ldap is developed and maintained by volunteers. We’re happy to share our work, and to work with you to make the library better, but (until you pay someone), there’s obligation to provide assistance.

So, keep it friendly, respectful, and supportive!

Mailing list

Discussion about the use and future of python-ldap occurs in the python-ldap@python.org mailing list.

It’s also the channel to use if documentation (including this guide) is not clear to you. Do try searching around before you ask on the list, though!

You can subscribe or unsubscribe to this list or browse the list archive.

Issues

Please report bugs, missing features and other issues to the bug tracker at GitHub. You will need a GitHub account for that.

If you prefer not to open a GitHub account, you’re always welcome to use the mailing list.

Security Contact

If you found a security issue that should not be discussed publicly, please e-mail the maintainer at pviktori@redhat.com. If required, write to coordinate a more secure channel.

All other communication should be public.

5.5.3 Contributing code

If you’re used to open-source Python development with Git, here’s the gist:

• git clone https://github.com/python-ldap/python-ldap
• Use GitHub for the bug tracker and pull requests.
• Run tests with tox; ignore Python interpreters you don’t have locally.

Or, if you prefer to avoid closed-source services:

• git clone https://pagure.io/python-ldap
• Send bug reports and patches to the mailing list.
• Run tests with tox; ignore Python interpreters you don’t have locally.
• Read the documentation directly at Read the Docs.

If you’re new to some aspect of the project, you’re welcome to use (or adapt) our sample workflow.
5.5.4 Additional tests and scripts

We use several specialized tools for debugging and maintenance.

Make targets

Make targets currently use the python3 executable. Specify a different one using, for example:

```
make PYTHON=/usr/local/bin/python
```

Notable targets are:

- **make autoformat** Automatically re-formats C and Python code to conform to Python style guides (PEP 7 and PEP 8). Note that no backups are made – please commit any other changes before using this target.
  
  Requires the indent program and the autopep8 Python module.

- **make lcov lcov-open** Generate and view test coverage for C code. Requires LCOV.

- **make scan-build** Run static analysis. Requires clang.

- **make valgrind** Run Valgrind to check for memory leaks. Requires valgrind and a Python suppression file, which you can specify as PYTHON_SUPP, e.g.:

  ```
  make valgrind PYTHON_SUPP=/your/path/to/valgrind-python.supp
  ```

  The suppression file is Misc/valgrind-python.supp in the Python source distribution, and it’s frequently packaged together with Python development headers.

Reference leak tests

Reference leak tests require a pydebug build of CPython and pytest with pytest-leaks plugin. A pydebug build has a global reference counter, which keeps track of all reference increments and decrements. The leak plugin runs each test multiple times and checks if the reference count increases.

Download and compile the pydebug build:

```
$ curl -O https://www.python.org/ftp/python/3.6.3/Python-3.6.3.tar.xz
$ tar xJf Python-3.6.3.tar.xz
$ cd Python-3.6.3
$ ./configure --with-pydebug
$ make
```

Create a virtual environment with the pydebug build:

```
$ ./python -m venv /tmp/refleak
$ /tmp/refleak/bin/pip install pytest pytest-leaks
```

Run reference leak tests:

```
$ cd path/to/python-ldap
$ /tmp/refleak/bin/pip install --upgrade .
$ /tmp/refleak/bin/pytest -v -R:
```

Run /tmp/refleak/bin/pip install --upgrade . every time a file outside of Tests/ is modified.
5.5.5 Instructions for core committers

If you have the authority (and responsibility) of merging changes from others, remember:

- All code changes need to be reviewed by someone other than the author.
- Tests must always pass. New features without tests shall not pass review.
- Make sure commit messages don’t use GitHub-specific link syntax. Use the full URL, e.g. https://github.com/python-ldap/python-ldap/issues/50 instead of #20.
  - Exception: it’s fine to use the short form in the summary line of a merge commit, if the full URL appears later.
  - It’s OK to use shortcuts in GitHub discussions, where they are not hashed into immutable history.
- Make a merge commit if the contribution contains several well-isolated separate commits with good descriptions. Use squash-and-merge (or fast-forward from a command line) for all other cases.
- It’s OK to push small changes into a pull request. If you do this, document what you have done (so the contributor can learn for the future), and get their ACK (confirmation) before merging.
- When squashing, do edit commit messages to add references to the pull request and relevant discussions/issues, and to conform to Git best practices.
  - Consider making the summary line suitable for the CHANGES document, and starting it with a prefix like Lib: or Tests:.
- Push to Pagure as well.

If you have good reason to break the “rules”, go ahead and break them, but mention why.

5.5.6 Instructions for release managers

If you are tasked with releasing python-ldap, remember to:

- Bump all instances of the version number.
- Go through all changes since last version, and add them to CHANGES.
- Run Additional tests and scripts as appropriate, fix any regressions.
- Change the release date in CHANGES.
- Merge all that (using pull requests).
- Run python setup.py sdist, and smoke-test the resulting package (install in a clean virtual environment, import ldap).
- Create Git tag python-ldap-{version}, and push it to GitHub and Pagure.
- Release the sdist on PyPI.
- Announce the release on the mailing list. Mention the Git hash.
- Add the release’s log from CHANGES on the GitHub release page.

5.6 python-ldap FAQ

5.6.1 Project

Q: Is python-ldap yet another abandon-ware project?
A1: “Jump on in.”
A2: “Jump into the C ;-)”
A3: see file CHANGES in source distribution or repository.

5.6.2 Usage

Q: Does it work with Python 3?
A0: Yes, from 3.0 on.
A1: For earlier versions, there’s pyldap, an independent fork now merged into python-ldap.

Q: Does it work with Python 2.6? (1.5|2.0|2.1|2.2|2.3|2.4|2.5)?
A: No. Old versions of python-ldap are still available from PyPI, though.

Q: My code imports module _ldap. That used to work, but after an upgrade it does not work anymore. Why?
A: Despite some outdated programming examples, the extension module _ldap MUST NOT be imported directly, unless you really know what you’re doing (e.g. for internal regression testing).

Import ldap instead, which is a Python wrapper around _ldap providing the full functionality.

Q: My script bound to MS Active Directory but a a search operation results in the exception ldap. OPERATIONS_ERROR with the diagnostic messages text “In order to perform this operation a successful bind must be completed on the connection.” What’s happening here?
A: When searching from the domain level, MS AD returns referrals (search continuations) for some objects to indicate to the client where to look for these objects. Client-chasing of referrals is a broken concept, since LDAPv3 does not specify which credentials to use when chasing the referral. Windows clients are supposed to simply use their Windows credentials, but this does not work in general when chasing referrals received from and pointing to arbitrary LDAP servers.

Therefore, per default, libldap automatically chases the referrals internally with an anonymous access which fails with MS AD.

So, the best thing to do is to switch this behaviour off:

```python
l = ldap.initialize('ldap://foobar')
l.set_option(ldap.OPT_REFERRALS, 0)
```

Q: Why am I seeing a ldap.SUCCESS traceback as output?
A: Most likely, you are using one of the non-synchronous calls, and probably mean to be using a synchronous call (see detailed explanation in Sending LDAP requests).

Q: Can I use LDAPv2 via python-ldap?
A: Yes, by explicitly setting the class attribute protocol_version.

You should not do that nowadays since LDAPv2 is considered historic since many years.

5.6.3 Installing

Q: Does it work with Windows 32?
A: Yes. You can find links to unofficial pre-compiled packages for Windows on the Installing python-ldap page.

Q: Can python-ldap be built against OpenLDAP 2.3 libs or older?
A: No. The needed minimal version of OpenLDAP is documented in Build prerequisites. Patched builds of python-ldap linked to older libs are not supported by the python-ldap project.

Q: During build there are warning messages displayed telling Lib/ldap.py and Lib/ldap/schema.py are not found:

```
warning: build_py: file Lib/ldap.py (for module ldap) not found
warning: build_py: file Lib/ldap/schema.py (for module ldap.schema) not found
```

A: ldap and ldap.schema are both module packages (directories containing various sub-modules). The messages above are falsely produced by DistUtils. Don’t worry about it.

Q: What’s the correct way to install on Mac OS X?

A:

```
xcode-select --install
pip install python-ldap
    --global-option=build_ext
    --global-option="-I$(xcrun --show-sdk-path)/usr/include/sasl"
```

Q: While importing module ldap, some shared lib files are not found. The error message looks similar to this:

```
ImportError: ld.so.1: /usr/local/bin/python: fatal: liblber.so.2: open failed: No such file or directory
```

A: You need to make sure that the path to liblber.so.2 and libldap.so.2 is in your LD_LIBRARY_PATH environment variable.

A2: Alternatively, if you’re on Linux, you can add the path to liblber.so.2 and libldap.so.2 to /etc/ld.so.conf and invoke the command ldconfig afterwards.

5.6.4 Historic

Q: Can python-ldap 2.x be built against Netscape, Mozilla or Novell libs?

A: Nope.

Q: My binary version of python-ldap was build with LDAP libs 3.3. But the python-ldap docs say LDAP libs 2.x are needed. I’m confused!

Short answer: See answer above and the Installing python-ldap page for a more recent version.

Long answer: E.g. some Win32 DLLs floating around for download are based on the old Umich LDAP code which is not maintained anymore for many years! Last Umich 3.3 release was 1997 if I remember correctly.

The OpenLDAP project took over the Umich code and started releasing OpenLDAP 1.x series mainly fixing bugs and doing some improvements to the database backend. Still, only LDAPv2 was supported at server and client side. (Many commercial vendors also derived their products from the Umich code.)

OpenLDAP 2.x is a full-fledged LDAPv3 implementation. It has its roots in Umich code but has many more features/improvements.

Q: While importing module ldap, there are undefined references reported. The error message looks similar to this:

```
ImportError: /usr/local/lib/libldap.so.2: undefined symbol: res_query
```

A: Especially on older Linux systems, you might have to explicitly link against libresolv.

Tweak setup.cfg to contain this line:
libs = lber ldap resolv
CHAPTER 6

Indices and tables

- genindex
- modindex
- search
Python Module Index

ldap (Posix, Windows), 14
ldap.asyncsearch, 28
ldap.controls, 31
ldap.controls.libldap, 33
ldap.controls.psearch, 34
ldap.controls.readentry, 34
ldap.controls.sessiontrack, 34
ldap.controls.simple, 32
ldap.dn, 34
ldap.extop, 36
ldap.extop.dds, 36
ldap.filter, 36
ldap.modlist, 37
ldap.resiter, 37
ldap.sas1, 43
ldap.schema, 38
ldap.schema.models, 40
ldap.schema.subentry, 38
ldap.syncrepl, 43
ldapuri, 46
ldif, 44

slapdtest, 49
Index

A

abandon() (ldap.LDAPObject method), 21
abandon_ext() (ldap.LDAPObject method), 22
add() (ldap.LDAPObject method), 22
add_ext() (ldap.LDAPObject method), 22
add_ext_s() (ldap.LDAPObject method), 22
add_s() (ldap.LDAPObject method), 22
addModlist() (in module ldap.modlist), 37
ADMINLIMIT_EXCEEDED, 18
AFFECTS_MULTIPLE_DSAS, 18
afterFirstResult() (ldap.asyncsearch.AsyncSearchHandler method), 29
ALIAS_DEREF_PROBLEM, 18
ALIAS_PROBLEM, 18
all_records (ldif.LDIFRecordList attribute), 45
allresults() (ldap.resiter.ResultProcessor method), 37
ALREADY_EXISTS, 18
applyDefaults() (ldapurl.LDAPUrl method), 47
AssertionControl (class in ldap.controls.libldap), 33
AsyncSearchHandler (class in ldap.asyncsearch), 29
attribute_types() (ldap.schema.models.Entry method), 40
attribute_types() (ldap.schema.subentry.SubSchema method), 39
AttributeType (class in ldap.schema.models), 40
AUTH_UNKNOWN, 18
AuthorizationIdentityRequestControl (class in ldap.controls.simple), 33
AuthorizationIdentityResponseControl (class in ldap.controls.simple), 33

B

bind() (ldap.LDAPObject method), 22
bind_s() (ldap.LDAPObject method), 22
BooleanControl (class in ldap.controls.simple), 32
BUSY, 18

C

callback() (ldap.sasl.sasl method), 43
cancel() (ldap.LDAPObject method), 22

CB_AUTHNAME (in module ldap.sasl), 43
CB_ECHOPROMPT (in module ldap.sasl), 43
CB_GETREALM (in module ldap.sasl), 43
CB_LANGUAGE (in module ldap.sasl), 43
CB_NOECHOPROMPT (in module ldap.sasl), 43
CB_PASS (in module ldap.sasl), 43
CB_USER (in module ldap.sasl), 43
CLIENT_LOOP, 18
cmpare() (ldap.LDAPObject method), 22
cmpare_ext() (ldap.LDAPObject method), 22
cmpare_ext_s() (ldap.LDAPObject method), 22
COMPARE_FALSE, 18
COMPARE_TRUE, 18
CONFIDENTIALITY_REQUIRED, 18
CONNECT_ERROR, 18
CONSTRAINT_VIOLATION, 18
CONTROL_NOT_FOUND, 18
cram_md5 (class in ldap.sasl), 44
CreateLDIF() (in module ldif), 44

D

DecodeControlTuples() (in module ldap.controls), 32
decodeControlValue() (ldap.controls.ResponseControl method), 31
decodeResponseValue() (ldap.extop.ExtendedResponse method), 36
DECODING_ERROR, 18
delete() (ldap.LDAPObject method), 22
delete_ext() (ldap.LDAPObject method), 23
delete_ext_s() (ldap.LDAPObject method), 23
delete_s() (ldap.LDAPObject method), 23
Dict (class in ldap.asyncsearch), 29
digest_md5 (class in ldap.sasl), 44
DITContentRule (class in ldap.schema.models), 41
DITStructureRule (class in ldap.schema.models), 42
dn2str() (in module ldap dn), 35
DN_FORMAT_AD_CANONICAL (in module ldap), 41
DN_FORMAT_DCE (in module ldap), 17
DN_FORMAT_LDAP (in module ldap), 17
LDAPBytesWarning (class in ldap), 21
LDAPControl (class in ldap.controls), 31
ldapdelete() (slapdtest.SlapdObject method), 49
LDAPError, 18
ldapmodify() (slapdtest.SlapdObject method), 49
LDAPObject (class in ldap), 21
LDAPUrl (class in ldapurl), 47
ldapurl (module), 46
ldapUrlEscape() (in module ldapurl), 47
LDAPUrlExtensions (class in ldapurl), 48
ldapwhoami() (slapdtest.SlapdObject method), 49
ldif (module), 44
LDIFCopy (class in ldif), 45
LDIFParser (class in ldif), 45
LDIFRecordList (class in ldif), 45
LDIFWriter (class in ldap.asyncsearch), 29
library_dirs (built-in variable), 11
libs (built-in variable), 11
List (class in ldap.asyncsearch), 29
listall() (ldap.schema.subentry.SubSchema method), 39
LOCAL ERROR, 19
LOOP_DETECT, 19

M
ManageDSAITControl (class in ldap.controls.simple), 32
MatchedValuesControl (class in ldap.controls.libldiap), 33
MatchingRule (class in ldap.schema.models), 41
MatchingRuleUse (class in ldap.schema.models), 41
modify() (ldap.LDAPObject method), 23
modify_ext() (ldap.LDAPObject method), 23
modify_ext_s() (ldap.LDAPObject method), 23
modify_s() (ldap.LDAPObject method), 23
modifyModlist() (in module ldap.modlist), 37
modrdn() (ldap.LDAPObject method), 23
modrdn_s() (ldap.LDAPObject method), 23
MORE_RESULTS_TO_RETURN, 19

N
NameForm (class in ldap.schema.models), 42
NAMING_VIOLATION, 19
NO_MEMORY, 19
NO_OBJECT_CLASS_MODS, 19
NO_RESULTS_RETURNED, 19
NO_SUCH_ATTRIBUTE, 19
NO_SUCH_OBJECT, 19
NOT_ALLOWED_ON_NONLEAF, 19
NOT_ALLOWED_ON_RDN, 19
NOT_HUMAN_READABLE_LDAP_SYNTAXES (in module ldap.schema.subentry), 38
NOT_SUPPORTED, 19

O
OBJECT_CLASS_VIOLATION, 19
ObjectClass (class in ldap.schema.models), 40
OctetStringInteger (class in ldap.controls.simple), 32
OPERATIONS_ERROR, 20
OPT_API_FEATURE_INFO (in module ldap), 15
OPT_API_INFO (in module ldap), 15
OPT_CLIENT_CONTROLS (in module ldap), 15
OPT_DEBUG_LEVEL (in module ldap), 15
OPT_DEFBASE (in module ldap), 15
OPT_DEREF (in module ldap), 16
OPT_DIAGNOSTIC_MESSAGE (in module ldap), 16
OPT_ERROR_STRING (in module ldap), 16
OPT_HOST_NAME (in module ldap), 16
OPT_MATCHED_DN (in module ldap), 16
OPT_NETWORK_TIMEOUT (in module ldap), 16
OPT_PROTOCOL_VERSION (in module ldap), 16
OPT_REFERRALS (in module ldap), 16
OPT_REFHOLIMIT (in module ldap), 16
OPT_RESTART (in module ldap), 16
OPT_SERVER_CONTROLS (in module ldap), 16
OPT_SIZELIMIT (in module ldap), 16
OPT_SUCCESS (in module ldap), 16
OPT_TIMELIMIT (in module ldap), 16
OPT_TIMEOUT (in module ldap), 16
OPT_URI (in module ldap), 16
OPT_X_KEEPALIVE_IDLE (in module ldap), 17
OPT_X_KEEPALIVE_INTERVAL (in module ldap), 17
OPT_X_KEEPALIVE_PROBES (in module ldap), 17
OPT_X_SASL_AUTHCID (in module ldap), 16
OPT_X_SASL_AUTHZID (in module ldap), 16
OPT_X_SASL_MEC("H (in module ldap), 16
OPT_X_SASL_REALM (in module ldap), 16
OPT_X_SASL_SECPRops (in module ldap), 16
OPT_X_SASL_SSL (in module ldap), 16
OPT_X_SASL_SSL_EXTERNAL (in module ldap), 16
OPT_X_SASL_SSL_MAX (in module ldap), 16
OPT_X_SASL_SSL_MIN (in module ldap), 16
OPT_X_TLS (in module ldap), 17
OPT_X_TLS_ALLOW (in module ldap), 17
OPT_X_TLS_CACERTDIR (in module ldap), 17
OPT_X_TLS_CIPHER_SUITE (in module ldap), 17
OPT_X_TLSCTX (in module ldap), 17
OPT_X_TLS_DEMAND (in module ldap), 17
OPT_X_TLS_HARD (in module ldap), 17
OPT_X_TLS_KEYFILE (in module ldap), 17
OPT_X_TLS_NEVER (in module ldap), 17
OPT_X_TLS_RANDOM_FILE (in module ldap), 17
OPT_X_TLS_REQUIRE_CERT (in module ldap), 17
OPT_X_TLS_TRY (in module ldap), 17
OTHER, 20
PARAM_ERROR, 20
parse() (ldif.LDIFParser method), 45
parse_entry_records() (ldif.LDIFParser method), 45
ParseLDIF() (in module ldif), 45
PARTIAL_RESULTS, 20
passwd() (ldap.LDAPObject method), 24
passwd_s() (ldap.LDAPObject method), 24
PORT (in module ldap), 15
postProcessing() (ldap.asyncrequest.AsyncSearchHandler method), 29
preProcessing() (ldap.asyncrequest.AsyncSearchHandler method), 29
processResults() (ldap.asyncrequest.AsyncSearchHandler method), 29
PROTOCOL_ERROR, 20
ProxyAuthzControl (class in ldap.controls.simple), 33
ReconnectLDAPObject (class in ldap), 21
RelaxRulesControl (class in ldap.controls.simple), 33
rename() (ldap.LDAPObject method), 24
rename_s() (ldap.LDAPObject method), 24
RequestControl (class in ldap.controls), 31
RequestControlTuples() (in module ldap.controls), 32
ResponseControl (class in ldap.controls), 31
Restart() (slapdtest.SlapdObject), 49
result() (ldap.LDAPObject method), 24
result2() (ldap.LDAPObject method), 25
result3() (ldap.LDAPObject method), 25
result4() (ldap.LDAPObject method), 25
ResultProcessor (class in ldap.resiter), 37
RESULTS_TOO_LARGE, 20
RFC
RFC 1779, 34
RFC 2589, 36
RFC 2696, 34
RFC 2830, 27
RFC 2849, 44
RFC 3062, 24
RFC 3296, 33
RFC 3829, 33
RFC 3876, 34
RFC 3909, 22
RFC 4370, 33
RFC 4422, 43
RFC 4512, 38
RFC 4513, 43
RFC 4514, 34
RFC 4515, 26, 36
RFC 4516, 15, 46
RFC 4527, 34
RFC 4528, 33
RFC 4532, 27
RFC 4533, 43
S
sasl (class in ldap.sasl), 43
SASL_AVAIL (in module ldap), 15
SASL_BIND_IN_PROGRESS, 20
sasl_external_bind_s() (ldap.LDAPObject method), 25
sasl_gssapi_bind_s() (ldap.LDAPObject method), 26
sasl_interactive_bind_s() (ldap.LDAPObject method), 25
sasl_non_interactive_bind_s() (ldap.LDAPObject method), 25
SchemaElement (class in ldap.schema.models), 40
search() (ldap.LDAPObject method), 26
search_ext() (ldap.LDAPObject method), 26
search_ext_s() (ldap.LDAPObject method), 26
search_s() (ldap.LDAPObject method), 26
SEARCH_SCOPE (in module ldapurl), 47
SEARCH_SCOPE_STR (in module ldapurl), 47
search_st() (ldap.LDAPObject method), 26
server_class (slapdtest.SlapdTestCase), 50
SERVER_DOWN, 20
set_option() (in module ldap), 15
set_option() (ldap.LDAPObject method), 27
setup_rundir() (slapdtest.SlapdObject), 49
simple_bind() (ldap.LDAPObject method), 26
simple_bind_s() (ldap.LDAPObject method), 26
SimpleLDAPObject (class in ldap), 21
SimplePagedResultsControl (class in ldap.controls), 34
SIZELIMIT_EXCEEDED, 20
SlapdObject (class in slapdtest), 49
slapdtest (module), 49
SlapdTestCase (class in slapdtest), 50
start() (slapdtest.SlapdObject), 49
start_tls_s() (ldap.LDAPObject method), 27
startSearch() (ldap.asyncrequest.AsyncSearchHandler method), 29
stop() (slapdtest.SlapdObject), 49
str2dn() (in module ldap.dn), 35
STRONG_AUTH_NOT_SUPPORTED, 20
STRONG_AUTH_REQUIRED, 20
SubSchema (class in ldap.schema.subentry), 38
T
TIMELIMIT_EXCEEDED, 20
TIMEOUT, 20
TLS_AVAIL (in module ldap), 15
tree() (ldap.schema.subentry/SubSchema method), 39
TYPE_OR_VALUE_EXISTS, 20
U
UNAVAILABLE, 20
UNAVAILABLE_CRITICAL_EXTENSION, 20
unbind() (ldap.LDAPObject method), 27
unbind_ext() (ldap.LDAPObject method), 27
unbind_ext_s() (ldap.LDAPObject method), 27
unbind_s() (ldap.LDAPObject method), 27
UNDEFINED_TYPE, 20
unparse() (ldapurl.LDAPUrl method), 48
unparse() (ldif.LDIFWriter method), 45
UNWILLING_TO_PERFORM, 20
urlfetch() (in module ldap.schema.subentry), 38
USER_CANCELED, 20

V
ValueLessRequestControl (class in ldap.controls.simple),
  32

W
wait() (slapdtest.SlapdObject method), 49
whoami_s() (ldap.LDAPObject method), 27