A low-level LDAP library for Python

David Leonard

February 1, 2002

The University of Queensland, Australia
E-mail: python-ldap-dev@lists.sourceforge.net

Abstract

This documents a simple Python C wrapper module for the OpenLDAP library.

CONTENTS

CHAPTER

ONE

The ldap module

1.1 `ldap` — LDAP library interface module

This module provides access to the LDAP (Lightweight Directory Access Protocol) C interface. It is similar to the interface described in RFC 1823, with the notable differences that lists are manipulated via Python list operations, and errors appear as exceptions.

For far more detailed information on the C interface, please see the documentation that accompanies the underlying LDAP Clibrary, or see RFC 1823.

This documentation is current for the Python LDAP module, version . Source and binaries are available from http://python-ldap.sourceforge.net/.

Much of this documentation has been derived from RFC 1823 and the manual pages accompanying the University of Michigan's original LDAP library implementation.

1.1.1 Functions

The `ldap` module defines the following functions:

`initialize`(*uri*)
Opens a new connection with an LDAP server, and return an LDAP object (see section 1.1.5) used to perform operations on that server. Parameter *uri* has to be a valid LDAP URL

See Also:

RFC 2255, "*The LDAP URL Format*"

`open`(*host* [, *port=PORT*])
Opens a new connection with an LDAP server, and return an LDAP object (see section 1.1.5) used to perform operations on that server. *host* is a string containing solely the host name. *port* is an integer specifying the port where the LDAP server is listening (default is 389). Note: Using this function is deprecated.

`dn2ufn`(*dn*)
Turns *dn* into a more user-friendly form, stripping off type names.

See Also:

RFC 1781, "*Using the Directory to Achieve User Friendly Naming*"

`explode_dn`(*dn* [, *notypes=0*])
This function takes *dn* and breaks it up into its component parts. Each part is known as an RDN (Relative Distinguished Name). The *notypes* parameter is used to specify that only the RDN values be returned and not their types. For example, the DN `"cn=Bob, c=US"` would be returned as either `["cn=Bob", "c=US"]` or `["Bob","US"]` depending on whether *notypes* was `0` or `1`, respectively.

See Also:

RFC 2253, "*Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*"

explode_rdn(*rdn* [, *notypes=0*])
This function takes a (multi-valued) *rdn* and breaks it up into a list of characteristic attributes. The *notypes* parameter is used to specify that only the RDN values be returned and not their types.

is_ldap_url(*url*)
This function returns true if *url* 'looks like' an LDAP URL (as opposed to some other kind of URL).

get_option(*option*)
This function returns the value of the global option specified by *option*.

set_option(*option, invalue*)
This function sets the value of the global option specified by *option* to *invalue*.

1.1.2 Constants

The module defines the following constants:

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

1.1.3 Exceptions

The module defines the following exceptions:

exception LDAPError
This is the base class of all execeptions raised by the module. Unlike the C interface, errors are not returned as result codes, but are instead turned into exceptions, raised as soon 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 ALIAS_DEREF_PROBLEM
A problem was encountered when dereferencing an alias. (Sets the `'matched'` field.)

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

exception ALREADY_EXISTS
The entry already exists.

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

exception BUSY
The DSA is busy.

exception COMPARE_FALSE
A compare operation returned false. (This exception should never be seen.)

exception COMPARE_TRUE
A compare operation returned true. (This exception should never be seen.)

exception CONSTRAINT_VIOLATION
An attribute value specified violates some constraint (e.g., a postalAddress has too many lines, or a line that is too long).

exception DECODING_ERROR
An error was encountered decoding a result from the LDAP server.

exception ENCODING_ERROR
An error was encountered encoding parameters to send to the LDAP server.

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

exception INAPPROPRIATE_AUTH
Inappropriate authentication was spec- ified (e.g., `LDAP_AUTH_SIMPLE` was specified and the entry does not have a userPassword attribute).

exception INAPPROPRIATE_MATCHING
Filter type not supported for the specified attribute.

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

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

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

exception INVALID_SYNTAX
An invalid attribute value was specified.

exception IS_LEAF
The object specified is a leaf of the diretcory tree. Sets the `'matched'` field of the exception dictionary value.

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

exception LOOP_DETECT
A loop was detected.

exception NAMING_VIOLATION
A naming violation occurred.

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

exception NOT_ALLOWED_ON_RDN
The operation is not allowed on an RDN.

exception NO_OBJECT_CLASS_MODS
Object class modifications are not allowed.

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

exception NO_SUCH_OBJECT
The specified object does not exist in the directory. Sets the `'matched'` field of the exception dictionary value.

exception OBJECT_CLASS_VIOLATION
An object class violation occurred (e.g., a ”must” attribute was missing from the entry).

exception OPERATIONS_ERROR
An operations error occurred.

exception OTHER
An unclassified error occurred.

exception PARAM_ERROR

An ldap routine was called with a bad parameter.

exception PARTIAL_RESULTS
Partial results only returned. (This exception should never be seen.)

exception PROTOCOL_ERROR
A violation of the LDAP protocol was detected.

exception RESULTS_TOO_LARGE
(Undocumented.)

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

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

exception STRONG_AUTH_NOT_SUPPORTED
The LDAP server does not support strong authentication.

exception STRONG_AUTH_REQUIRED
Strong authentication is required for the operation.

exception TIMELIMIT_EXCEEDED
An LDAP time limit was exceeded.

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

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

exception UNAVAILABLE
The DSA is unavailable.

exception UNDEFINED_TYPE
(Undocumented.)

exception UNWILLING_TO_PERFORM
The DSA is unwilling to perform the operation.

exception 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.

1.1.4 Example

The following example demonstrates how to open an LDAP server using the `ldap` module.

```
>>> import ldap
>>> l = ldap.initialize("ldap://my-ldap-server.my-domain:389")
>>> l.simple_bind_s("","")
>>> l.search_s("o=My Organisation, c=AU", ldap.SCOPE_SUBTREE, "objectclass=*")
```

1.1.5 LDAP Objects

LDAP objects are returned by `open()` when a successful connection to an LDAP server is established. The connection is automatically unbound and closed when the LDAP object is deleted.

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. See below for a description of the `result()` method which describes the data structure returned from the server.

LDAP objects, have the following methods:

`fileno``()`
: Return the file descriptor associated with the connection, for use with the `select` module. (This method is available only if the underlying library can supply it.)

LDAP operations

`abandon``(`*msgid*`)`
: Abandons or cancels an LDAP operation in progress. 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()`.

`add``(`*dn, modlist*`)`
`add_s``(`*dn, modlist*`)`
: This function is similar to `modify()`, except that the operation integer is omitted from the tuples in *modlist*.

`bind``(`*who, cred, method*`)`
`bind_s``(`*who, cred, method*`)`
`simple_bind``(`*who, passwd*`)`
`simple_bind_s``(`*who, passwd*`)`
: 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`.

`compare``(`*dn, attr, value*`)`
`compare_s``(`*dn, attr, value*`)`
: Perform an LDAP comparison between the attribute named *attr* of entry *dn*, and the value *value*. The synchronous form returns `0` for false, or `1` for true. The asynchronous form returns the message ID of the initiated request, and the result of the asynchronous compare can be obtained using `result()`.

  Note that the asynchronous technique yields the answer by raising the exception objects `COMPARE_TRUE` or `COMPARE_FALSE`.

  Note A design fault in the LDAP API prevents *value* from containing nul characters.

`delete``(`*dn*`)`
`delete_s``(`*dn*`)`
: 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()`.

`modify``(` *dn, modlist* `)`
`modify_s``(` *dn, modlist* `)`
: 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 `MOD_ADD`, `MOD_DELETE`, or `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.

  The asynchronous method `modify()` returns the message ID of the initiated request.

modrdn(*dn, newrdn* [, *delold=1*])
modrdn_s(*dn, newrdn* [, *delold=1*])

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. (Note: This may not actually be supported by the underlying library.) The asynchronous version returns the initiated message id.

rename(*dn, newrdn* [, *newsuperior=None*] [, *delold=1*])
rename_s(*dn, newrdn* [, *newsuperior=None* [, *delold=1*]])

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.

result([*msgid=RES_ANY* [, *all=1* [, *timeout=-1*]]])

This method is used to wait for and return the result of an operation previously initiated by one of the LDAP *asynchronous* operations (eg `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`.

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

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 `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: `'RES_BIND'`, `'RES_SEARCH_ENTRY'`, `'RES_SEARCH_RESULT'`, `'RES_MODIFY'`, `'RES_ADD'`, `'RES_DELETE'`, `'RES_MODRDN'`, or `'RES_COMPARE'`. (The module constants `RES_*` are set to these strings, for your convenience.)

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.

search(*base, scope, filter* [, *attrlist=None* [, *attrsonly=0*]])
search_s(*base, scope, filter* [, *attrlist=None* [, *attrsonly=0*]])
search_st(*base, scope, filter* [, *attrlist=None* [, *attrsonly=0* [, *timeout=-1*]]])

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 *filter* argument is a string representation of the filter to apply in the search. Simple filters can be specified as "*attribute_type=attribute_value*".

See Also:

RFC 2254, "*The String Representation of LDAP Search Filters*"

When using the asynchronous form and `result()`, the *all* parameter affects how results come in. For *all* set to `0`, result tuples trickle in (with the same message id), and with the result type `RES_SEARCH_ENTRY`, 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.

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 extracted using the underlying `ldap_get_dn()` function, 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.

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

`set_rebind_proc`(*func*)
If a referral is returned from the server, automatic re-binding can be achieved by providing a function that accepts as an argument the newly opened LDAP object and returns the tuple `(who, cred, method)`.

Passing a value of `None` for *func* will disable this facility.

Because of restrictions in the implementation, only one rebinding function is supported at any one time. In addition, this method is only available if support is available in the underlying library (LDAP_REFERRALS).

`unbind`()
`unbind_s`()
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.

The `unbind()` and `unbind_s()` methods are both synchronous in nature

`url_search_s`(*url* [, *attrsonly=0*])
`url_search_s`(*url* [, *attrsonly=0* [, *timeout=-1*]])
These routine works much like the `search()` family of methods, except that many of the search parameters are extracted from the *url* argument.

LDAP URLs are fully described in RFC 2255.

LDAP options

`get_option`(*option*)
This function returns the value of the LDAPObject option specified by *option*.

`set_option`(*option, invalue*)
This function sets the value of the LDAPObject option specified by *option* to *invalue*.

Cache control

The following methods are available only if the underlying library supports a local cache and cache control.

`destroy_cache`()
Turns off caching and removed it from memory.

`disable_cache`()
Temporarily disables use of the cache. New requests are not cached, and the cache is not checked when returning results. Cache contents are not deleted.

`enable_cache`([*timeout=NO_LIMIT*, [*maxmem=NO_LIMIT*]])
Using a cache often greatly improves performance. By default the cache is disabled. Specifying *timeout* in seconds is used to decide how long to keep cached requests. The *maxmem* value is in bytes, and is used to set

an upper bound on how much memory the cache will use. A value of NO_LIMIT for either indicates unlimited. Subsequent calls to enable_cache() can be used to adjust these parameters.

This and other caching methods are not available if the library and the ldap module were compiled with -DNO_CACHE.

flush_cache()
Deletes the cache's contents, but does not affect it in any other way.

set_cache_options(*option*)
Changes the caching behaviour. Currently supported options are CACHE_OPT_CACHENOERRS, which suppresses caching of requests that resulted in an error, and CACHE_OPT_CACHEALLERRS, which enables caching of all requests. The default behaviour is not to cache requests that result in errors, except those that result in a SIZELIMIT_EXCEEDED exception.

uncache_entry(*dn*)
Removes all cached entries that make reference to *dn*. This should be used, for example, after doing a modify() involving *dn*.

uncache_request(*msgid*)
Remove the request indicated by *msgid* from the cache.

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.

deref
Controls for when an automatic dereference of a referral occurs. This must be one of DEREF_NEVER, DEREF_SEARCHING, DEREF_FINDING, or DEREF_ALWAYS.

errno
error
matched
These attributes are set after an exception has been raised, and are also included with the value raised. (See also section 1.1.3.) (All read-only.)

lberoptions
Options for the BER library.

options
General options. This field is the bitiwse OR of the flags OPT_REFERRALS (follow referrals), and OPT_RESTART (restart the select() system call when interrupted).

refhoplimit
Maximum number of referrals to follow before raising an exception. Defaults to 5.

sizelimit
Limit on size of message to receive from server. Defaults to NO_LIMIT.

timelimit
Limit on waiting for any response, in seconds. Defaults to NO_LIMIT.

valid
If zero, the connection has been unbound. See unbind() for more information. (read-only)

INDEX