Python LDAP module reference

David Leonard

July 13, 1998

0.1 Module `ldap`

This module provides access to the University of Michigan's LDAP (Lightweight Directory Access Protocol) C interface. It is more-or-less compliant with the interface described in RFC 1823, with the notable differences that lists are manipulated via Python list operations, and errors appear as exceptions.

For more detailed information on the C interface, please see the documentation that accompanies the package available from `ftp://terminator.rs.itd.umich.edu/ldap/ldap-3.3.tar.Z`.

This documentation is current for the python ldap module, version 1.4.

0.1.1 Constants and Exceptions

The following exceptions and constants are exported from the module:

`error`
: This (generic) exception is raised when some other error arises within the glue code between the C interface and the Python API.

`ALIAS_DEREF_PROBLEM`
`ALIAS_PROBLEM`
`ALREADY_EXISTS`
`AUTH_UNKNOWN`
`BUSY`
`COMPARE_FALSE`
`COMPARE_TRUE`
`CONSTRAINT_VIOLATION`
`DECODING_ERROR`
`ENCODING_ERROR`
`FILTER_ERROR`
`INAPPROPRIATE_AUTH`
`INAPPROPRIATE_MATCHING`
`INSUFFICIENT_ACCESS`
`INVALID_CREDENTIALS`
`INVALID_DN_SYNTAX`
`INVALID_SYNTAX`
`IS_LEAF`
`LOCAL_ERROR`
`LOOP_DETECT`

NAMING_VIOLATION
NOT_ALLOWED_ON_NONLEAF
NOT_ALLOWED_ON_RDN
NO_OBJECT_CLASS_MODS
NO_SUCH_ATTRIBUTE
NO_SUCH_OBJECT
OBJECT_CLASS_VIOLATION
OPERATIONS_ERROR
OTHER
PARAM_ERROR
PARTIAL_RESULTS
PROTOCOL_ERROR
RESULTS_TOO_LARGE
SERVER_DOWN
SIZELIMIT_EXCEEDED
STRONG_AUTH_NOT_SUPPORTED
STRONG_AUTH_REQUIRED
TIMELIMIT_EXCEEDED
TIMEOUT
TYPE_OR_VALUE_EXISTS
UNAVAILABLE
UNDEFINED_TYPE
UNWILLING_TO_PERFORM
USER_CANCELLED

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

`PORT`

The standard TCP port that LDAP servers listen on.

Many other (undocumented) constants available in the module correspond to those found in the LDAP header file, `<ldap.h>`.

0.1.2 Functions

The following functions are available at the module level:

`dn2ufn(`*dn*`)`

Turns the DN *dn* into a more user-friendly form, stripping off type names. See RFC 1781 “Using the Directory to Achieve User Friendly Naming” for more details on the UFN format.

`explode_dn`(*dn* [, *notypes*=`0`])
: This function takes the DN *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.

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

`open`(*host* [, *port*=`PORT`])
: Opens a new connection with an LDAP server, and returns an LDAP object representative of this.

0.1.3 LDAP Objects

LDAP objects are created by the `open()` function defined at the top level of the module. The connection is automatically unbound and closed during garbage collection.

Most methods initiate an asynchronous request to the LDAP server and return a message id that can be used later to retrieve the result. The methods ending with `_s` are the synchronous form and wait for and return with the server's result, *á la* `result()`, or with `None` if no data is expected. See the `result()` method for a description of the data structure returned from the server.

Exceptions from methods

Unlike the C library, 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 containing extra information.

This dictionary contains an entry for the key `'desc'` for an English description of the error class and `'info'` which contains a string containing more information the server may have sent.

If the exception was one of `NO_SUCH_OBJECT`, `ALIAS_PROBLEM`, `INVALID_DNS_SYNTAX`, `IS_LEAF`, or `ALIAS_DEREFERENCING_PROBLEM`, then `'matched'` will be a key for the name of the lowest entry (object or alias) that was matched and is a truncated form of the name provided or aliased dereferenced.

Methods on LDAP Objects

`abandon(`*msgid*`)`

Abandons or cancels an LDAP operation in progress. The *msgid* 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 no operation integer need be included in the tuples.

`bind(`*who*`,` *cred*`,` *method*`)`
`bind_s(`*who*`,` *cred*`,` *method*`)`
`simple_bind(`*who*`,` *passwd*`)`
`simple_bind_s(`*who*`,` *passwd*`)`
`kerberos_bind_s(`*who*`)`
`kerberos_bind1(`*who*`)`
`kerberos_bind1_s(`*who*`)`
`kerberos_bind2(`*who*`)`
`kerberos_bind2_s(`*who*`)`

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. The general method `bind()` takes a third parameter, *method* which can be one of `AUTH_SIMPLE`, `AUTH_KRBV41` or `AUTH_KRBV42`. The *cred* parameter is ignored for Kerberos authentication.

Kerberos authentication is only available if the LDAP library and the ldap module were compiled with `-DWITH_KERBEROS`.

`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 initiates request, and the result of the asynchronous compare can be obtained using `result()`.

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

A design bug in the library 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()`.

`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.

`modify(` *dn*`,` *modlist* `)`
`modify_s(` *dn*`,` *modlist* `)`

Performs an LDAP modify operation on an entry's attributes. *dn* is the DN of the entry to modify, and *modlist* is the list of modifications to make to the entry.

Each element of the list *modlist* should be a tuple of the form `(mod_op,mod_type,mod_vals)`, where *mod_op* is 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 `modify()` returns the message id of the initiated request.

See the `ber_*` methods for decoding Basic Encoded ASN.1 values. (To be implemented.)

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

Perform a modify RDN 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 actually corresponds to the `modrdn2*` routines in the C library.

`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 operation routines (eg `search()`, `modify()`, etc.) They all returned an invocation identifier (a message id) upon successful initiation of their operation. This id is guaranteed to be unique across an LDAP session, and can be used to request the result of a specific operation via the *msgid* parameter of the `result()` method.

If the result of a specific operation is required, *msgid* should be set to the invocation message id returned when the operation was initiated; otherwise `RES_ANY` should be supplied.

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, ie after all entries and the final search result have been received.

The method returns a tuple of the form `(`*result_type*`,` *result_data*`)`. The *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 constants `RES_*` are set to these strings, for convenience.

See `search()` for a description of the search result's *result_data*, otherwise the *result_data* is normally meaningless.

The `result()` method will block for *timeout* seconds, or indefinitely if *timeout* is negative. A timeout of `0` will effect a poll. The timeout can be expressed as a floating-point value.

If a timeout occurs, the tuple `(None,None)` is returned.

`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).

filter is a string representation of the filter to apply in the search. Simple filters can be specified as `"`*attribute_type=attribute_value*`"`. More complex filters are specified using a prefix notation according to the following BNF:

filter	::=	`"("` *filtercomp* `")"`
filtercomp	::=	*and* | *or* | *not* | *simple*
and	::=	`"&"` *filterlist*
or	::=	`"|"` *filterlist*
not	::=	`"!"` *filter*
filterlist	::=	*filter* | *filter filterlist*
simple	::=	*attributetype filtertype attributevalue*
filtertype	::=	`"="` | `"~="` | `"<="` | `">="`

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()`, which may raise an exception of 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 time.

`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.

`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. This method is only available if the module and library were compiled with `-DLDAP_REFERRALS`.

`ufn_setfilter(`*filtername*`)`
`ufn_setprefix(`*prefix*`)`
`ufn_search_s(`*url* [, *attrsonly*=`0`] `)`
`ufn_search_st(`*url* [, *attrsonly*=`0` [, *timeout*=`-1`]] `)`
: See the LDAP library manual pages for more information on these 'user-friendly name' functions.

`unbind_s()`
`unbind()`
: 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 invalid. Further invocation of methods on the object will yield an exception.

The `unbind` and `unbind_s` methods are identical, and are synchronous in nature `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.

`url_search_s(`*url*`,` [*attrsonly*`=0`] `)`
`url_search_s(`*url*`,` [*attrsonly*`=0` [`,` *timeout*`=-1`]] `)`

These routine works much like `search_s*`, except that many search parameters are pulled out of the URL *url*.

LDAP URLs look like this:

`"ldap://`*host* `[:`*port*`] /`*dn* `[?`*attributes* `[?`*scope* `[?`*filter*`] ] ]"`

where *scope* is one of `base` (default), `one` or `sub`, and *attributes* is a comma-separated list of attributes to be retrieved.

URLs wrapped in angle-brackets and/or preceded by `"URL:"` are tolerated.

Attributes on LDAP Objects

Each LDAP object also sports the following attributes.

`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 read-only attributes are set after an exception has been raised, and are also included with the value raised. See the section 'Exceptions from methods', above.

`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. Defaults to `NO_LIMIT`.

`valid`
: If zero, the connection has been unbound. See `unbind()` for more information.