python-ldap Documentation

Release 2.3.11.0

python-ldap project

April 19, 2012

CONTENTS

Abstract

This document describes the package python-ldap with its various modules. This manual assumes basic knowledge about the Python language and the LDAP standard.

CHAPTER

ONE

CONTENTS

1.1 Building and installing

1.1.1 Prerequisites

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

- Python version 2.3 or later including its development files: http://www.python.org/
- OpenLDAP client libs version 2.3 or later: http://www.openldap.org/ It is not possible and not supported to build with prior versions.
- OpenSSL (optional): http://www.openssl.org/
- cyrus-sasl (optional): http://asg.web.cmu.edu/sasl/sasl-library.html
- Kerberos libs, MIT or heimdal (optional)

1.1.2 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 the documentation of Python's DistUtils.

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 *Libs used*).

extra_compile_args
: Compiler options.

extra_objects

Libs 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 if needed and present during build

ssl
The SSL/TLS library of OpenSSL if needed and present during build

crypto
The basic cryptographic library of OpenSSL if needed and present during build

1.1.3 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-2.3) and SASL header files found in /usr/include/sasl. Debugging symbols are preserved with compile option -g.

```
[_ldap]
library_dirs = /opt/openldap-2.3/lib
include_dirs = /opt/openldap-2.3/include /usr/include/sasl

extra_compile_args = -g
extra_objects =

libs = ldap_r lber sasl2 ssl crypto
```

1.2 `ldap` LDAP library interface module

Module author: python-ldap project (see http://www.python-ldap.org/)

This module provides access to the LDAP (Lightweight Directory Access Protocol) C API implemented in OpenLDAP 2.3 or newer. It is similar to the C API, 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 (expired) draft-ietf-ldapext-ldap-c-api-04. This documentation is current for the Python LDAP module, version 2.3.11.0. Source and binaries are available from http://www.python-ldap.org/.

1.2.1 Functions

The `ldap` module defines the following functions:

`ldap.`**`initialize`**(*uri*[, *trace_level=0*[, *trace_file=sys.stdout*[, *trace_stack_limit=None*]]])

Opens a new connection with an LDAP server, and return an LDAP object (see *LDAPObject class*) used to perform operations on that server. Parameter *uri* has to be a valid LDAP URL. The 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. 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 `3` for also logging the traceback of method calls.

See Also:

RFC 4516 - Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator

`ldap.`**`open`**(*host*[, *port=PORT*])

Opens a new connection with an LDAP server, and return an LDAP object (see *LDAPObject class*) 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.

`ldap.`**`get_option`**(*option*)

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

`ldap.`**`set_option`**(*option*, *invalue*)

This function sets the value of the global option specified by *option* to *invalue*.

1.2.2 Constants

The module defines various constants.

General

`ldap.`**`PORT`**

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

`ldap.`**`SASL_AVAIL`**

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

`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_options{3}*

For use with functions and method set_option() and get_option() the following option identifiers are defined as constants:

ldap.**OPT_API_FEATURE_INFO**

ldap.**OPT_API_INFO**

ldap.**OPT_CLIENT_CONTROLS**

ldap.**OPT_DEBUG_LEVEL**
Sets the debug level within the underlying LDAP C lib.

ldap.**OPT_DEREF**
Specifies how alias derefencing 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**

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**

ldap.**OPT_URI**

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**

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**

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**

ldap.**DN_P_NOSPACEAFTERRDN**

ldap.**DN_PEDANTIC**

1.2.3 Exceptions

The module defines the following exceptions:

exception ldap.**LDAPError**
This is the base class of all execeptions 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 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.**COMPARE_FALSE**
A compare operation returned false. (This exception should never be seen because compare() returns a boolean result.)

exception ldap.**COMPARE_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 diretcory 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 ldap.**LOOP_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.

1.2.4 LDAPObject class

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

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 `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 `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*)

`LDAPObject.`**`abandon_ext`**(*msgid*[, *serverctrls=None*[, *clientctrls=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 above.

`LDAPObject.`**`add`**(*dn*, *modlist*)

`LDAPObject.`**`add_s`**(*dn*, *modlist*)

`LDAPObject.`**`add_ext`**(*dn*, *modlist*[, *serverctrls=None*[, *clientctrls=None*]])

`LDAPObject.`**`add_ext_s`**(*dn*, *modlist*[, *serverctrls=None*[, *clientctrls=None*]])
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.

serverctrls and *clientctrls* like described above.

`LDAPObject.`**`bind`**(*who*, *cred*, *method*)

`LDAPObject.`**`bind_s`**(*who*, *cred*, *method*)

`LDAPObject.`**`simple_bind`**([*who=''*[, *cred=''*]])

LDAPObject.**simple_bind_s**([*who=''*[, *cred=''*]])
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.**sasl_interactive_bind_s**(*who*, *auth*)
This call is used to bind to the directory with a SASL bind request.

LDAPObject.**cancel**(*cancelid*[, *serverctrls=None*[, *clientctrls=None*]])
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 above.

RFC 3909 - Lightweight Directory Access Protocol (LDAP): Cancel Operation

LDAPObject.**compare**(*dn*, *attr*, *value*)

LDAPObject.**compare_s**(*dn*, *attr*, *value*)

LDAPObject.**compare_ext**(*dn*, *attr*, *value*[, *serverctrls=None*[, *clientctrls=None*]])

LDAPObject.**compare_ext_s**(*dn*, *attr*, *value*[, *serverctrls=None*[, *clientctrls=None*]])
Perform an LDAP comparison between the attribute named *attr* of entry *dn*, and the value *value*. The synchronous forms returns `0` for false, or `1` for true. The asynchronous forms 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 `ldap.COMPARE_TRUE` or `ldap.COMPARE_FALSE`.

serverctrls and *clientctrls* like described above.

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

LDAPObject.**delete**(*dn*)

LDAPObject.**delete_s**(*dn*)

LDAPObject.**delete_ext**(*dn*[, *serverctrls=None*[, *clientctrls=None*]])

LDAPObject.**delete_ext_s**(*dn*[, *serverctrls=None*[, *clientctrls=None*]])
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 above.

LDAPObject.**modify**(*dn*, *modlist*)

LDAPObject.**modify_s**(*dn*, *modlist*)

LDAPObject.**modify_ext**(*dn*, *modlist*[, *serverctrls=None*[, *clientctrls=None*]])

LDAPObject.**modify_ext_s**(*dn*, *modlist*[, *serverctrls=None*[, *clientctrls=None*]])

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.

serverctrls and *clientctrls* like described above.

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*.

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

LDAPObject.**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. The asynchronous version returns the initiated message id.

This operation is emulated by `rename()` and `rename_s()` methods since the mod-rdn2* routines in the C library are deprecated.

LDAPObject.**passwd**(*user*, *oldpw*, *newpw*[, *serverctrls=None*[, *clientctrls=None*]])

LDAPObject.**passwd_s**(*user*, *oldpw*, *newpw*[, *serverctrls=None*[, *clientctrls=None*]])

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.

serverctrls and *clientctrls* like described above.

The asynchronous version returns the initiated message id.

See Also:

RFC 3062 - LDAP Password Modify Extended Operation

LDAPObject.**rename**(*dn*, *newrdn*[, *newsuperior=None*[, *delold=1*[, *serverctrls=None*[, *clientctrls=None*]]]])

LDAPObject.**rename_s**(*dn*, *newrdn*[, *newsuperior=None*[, *delold=1*[, *serverctrls=None*[, *clientctrls=None*]]]])

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 above.

LDAPObject.**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` 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.

`LDAPObject.`**`result2`**`(`[*msgid=RES_ANY*[, *all=1*[, *timeout=-1*]]])

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.

`LDAPObject.`**`result3`**`(`[*msgid=RES_ANY*[, *all=1*[, *timeout=-1*]]])

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

`LDAPObject.`**`search`**`(`*base*, *scope*[, *filterstr='(objectClass=*)'*[, *attrlist=None*[, *attrsonly=0*]]])

`LDAPObject.`**`search_s`**`(`*base*, *scope*[, *filterstr='(objectClass=*)'*[, *attrlist=None*[, *attrsonly=0*]]])

`LDAPObject.`**`search_st`**`(`*base*, *scope*[, *filterstr='(objectClass=*)'*[, *attrlist=None*[, *attrsonly=0*[, *timeout=-1*]]]])

`LDAPObject.`**`search_ext`**`(`*base*, *scope*[, *filterstr='(objectClass=*)'*[, *attrlist=None*[, *attrsonly=0*[, *serverctrls=None*[, *clientctrls=None*[, *timeout=-1*[, *sizelimit=0*]]]]]]])

`LDAPObject.`**`search_ext_s`**`(`*base*, *scope*[, *filterstr='(objectClass=*)'*[, *attrlist=None*[, *attrsonly=0*[, *serverctrls=None*[, *clientctrls=None*[, *timeout=-1*[, *sizelimit=0*]]]]]]])

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:

RFC 4515 - Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters.

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 above.

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.

`LDAPObject.`**`start_tls_s`**`()`

See Also:

RFC 2830 - Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security

`LDAPObject.`**`unbind`**`()`

`LDAPObject.`**`unbind_s`**`()`

`LDAPObject.`**`unbind_ext`**`(`[*serverctrls=None*[, *clientctrls=None*]])

`LDAPObject.`**`unbind_ext_s`**`(`[*serverctrls=None*[, *clientctrls=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 above.

These methods are all synchronous in nature.

`LDAPObject.`**`whoami_s`**`()`

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:

RFC 4532 - Lightweight Directory Access Protocol (LDAP) "Who am I?" Operation

Connection-specific LDAP options

`LDAPObject.`**`get_option`**`(`*option*`)`

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

`LDAPObject.`**`set_option`**`(`*option*, *invalue*`)`

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`**
: 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 lib.

`LDAPObject.`**`network_timeout`**
: 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 lib.

`LDAPObject.`**`protocol_version`**
: 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 lib.

Note: It is highly recommended to set the protocol version after establishing a LDAP connection with `initialize()` and before submitting the first request.

`LDAPObject.`**`sizelimit`**
: 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 lib. Its use is deprecated in favour of *sizelimit* parameter when using `search_ext()`.

`LDAPObject.`**`timelimit`**
: 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 lib. Its use is deprecated in favour of using *timeout*.

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

1.2.5 Example

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

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

```
>>>   print 'Processing',repr(dn)
>>>   handle_ldap_entry(entry)
```

1.3 ldap.async Framework for stream-processing of large search results

Module author: python-ldap project (see http://www.python-ldap.org/)

1.3.1 Examples for ldap.async

Using ldap.async.List

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

```
import sys,ldap,ldap.async

s = ldap.async.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.\n')
else:
  if partial:
    sys.stderr.write('Warning: Only partial results received.\n')

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

Using ldap.async.LDIFWriter

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

```
import sys,ldap,ldap.async

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

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.\n')
else:
  if partial:
    sys.stderr.write('Warning: Only partial results received.\n')

sys.stderr.write(
  '%d results received.\n' % (
    s.endResultBreak-s.beginResultsDropped
  )
)
```

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

Module author: python-ldap project (see http://www.python-ldap.org/)

1.4.1 Examples for ldap.resiter

Using ldap.resiter

This example demonstrates how to use mix-in class ldap.resiter.ResultProcessor for retrieving results and processing them in a for-loop.

```
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 self.source.allresults(msg_id):
  for dn,entry in res_data:
    # process dn and entry
    print dn,entry['objectClass']
```

1.5 ldap.controls High-level access to LDAP controls

Module author: python-ldap project (see http://www.python-ldap.org/)

The `ldap.controls` module defines the following classes:

class `ldap.controls.`**LDAPControl**(*controlType*, *criticality*[, *controlValue=:const:'None'*[, *encodedControlValue=:const:'None'*]])

Base class for all LDAP controls. This class should not be used directly, instead one of the following subclasses should be used as appropriate.

encodeControlValue(*value*)

Dummy method to be overridden by subclasses.

decodeControlValue(*value*)

Dummy method to be overridden by subclasses.

getEncodedTuple()

Return a readily encoded 3-tuple which can be directly passed to C module _ldap. This method is called by function `ldap.EncodeControlTuples()`.

class `ldap.controls.`**BooleanControl**(*controlType*, *criticality*[, *controlValue=:const:'None'*[, *encodedControlValue=:const:'None'*]])

Base class for simple controls with booelan control value. In this base class *controlValue* has to be passed as boolean type (`True`/`False` or `1`/`0`).

class `ldap.controls.`**SimplePagedResultsControl**(*controlType*, *criticality*[, *controlValue=:const:'None'*[, *encodedControlValue=:const:'None'*]])

The class provides the LDAP Control Extension for Simple Paged Results Manipulation. *controlType* is ignored in favor of `ldap.LDAP_CONTROL_PAGE_OID`.

See Also:

RFC 2696 - LDAP Control Extension for Simple Paged Results Manipulation

class ldap.controls.**MatchedValuesControl**(*criticality*[, *controlValue=:const:'None'*])

This class provides the LDAP Matched Values control. *controlValue* is an LDAP filter.

See Also:

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

The ldap.controls module defines the following functions:

ldap.controls.**EncodeControlTuples**(*ldapControls*)

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

ldap.controls.**DecodeControlTuples**(*ldapControlTuples*)

Decodes a list of readily encoded 3-tuples as returned by the C module _ldap.

1.6 ldap.dn LDAP Distinguished Name handling

Module author: python-ldap project (see http://www.python-ldap.org/)

See Also:

For LDAPv3 DN syntax see:

RFC 4514 - Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names For 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*)

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=:const:'0'*])

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*).

ldap.dn.**dn2str**(*dn*)

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=:const:'0'*[, *flags=:const:'0'*]])

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=:const:'0'*[, *flags=:const:'0'*]])

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.

1.6.1 Examples

Splitting a LDAPv3 DN to AVA level:

```
>>> ldap.dn.str2dn('cn=Michael Str\xc3\xb6der,dc=stroeder,dc=com',flags=ldap.DN_
[[('cn', 'Michael Str\xc3\xb6der', 4)], [('dc', 'stroeder', 1)], [('dc', 'com',
```

Splitting a LDAPv2 DN into RDN parts:

```
>>> ldap.dn.explode_dn('cn=Michael Stroeder;dc=stroeder;dc=com',flags=ldap.DN_FO
['cn=Michael Stroeder', 'dc=stroeder', 'dc=com']
```

Splitting a multi-valued RDN:

```
>>> ldap.dn.explode_rdn('cn=Michael Stroeder+mail=michael@stroeder.com',flags=ld
['cn=Michael Stroeder', 'mail=michael@stroeder.com']
```

Splitting a LDAPv3 DN with a multi-valued RDN into its AVA parts:

```
>>> ldap.dn.str2dn('cn=Michael Stroeder+mail=michael@stroeder.com,dc=stroeder,dc
[[('cn', 'Michael Stroeder', 1), ('mail', 'michael@stroeder.com', 1)], [('dc', '
```

1.7 `ldap.filter` LDAP filter handling

Module author: python-ldap project (see http://www.python-ldap.org/)

See Also:

RFC 4515 - Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters.

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

`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.

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.

1.8 ldap.modlist Generate modify lists

Module author: python-ldap project (see http://www.python-ldap.org/)

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

ldap.modlist.**addModlist**(*entry*[, *ignore_attr_types*=[]])
This function builds a list suitable for passing it directly as argument *modlist* to method `add()` or its synchronous counterpart `add_s()`. *entry* is a dictionary like returned when receiving search results.

ldap.modlist.**modifyModlist**(*old_entry*, *new_entry*[, *ignore_attr_types=[]*[, *ignore_oldexistent=0*]])
This function builds a list suitable for passing it directly as argument *modlist* to method `modify()` or its synchronous counterpart `modify_s()`. Roughly when applying the resulting modify list to an entry holding the data *old_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. *old_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. If *ignore_oldexistent* is non-zero attribute type names which are in *old_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 '' for deleting an attribute. In most cases leave zero.

1.9 ldap.schema Processing LDAPv3 sub schema sub entry

Module author: python-ldap project (see http://www.python-ldap.org/)

1.9.1 Examples for ldap.schema

```
import ldap.schema
```

1.10 ldif LDIF parser and generator

Module author: python-ldap project (see http://www.python-ldap.org/)

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

1.10.1 Example

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

```
>>> import sys,ldif
>>> entry={'objectClass':['top','person'],'cn':['Michael Stroeder'],'sn':['Stroe
>>> 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.

```
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):
      for i in skip_dn:
         if i == dn: return
      self.writer.unparse(dn, entry)

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

1.11 `ldapurl` LDAP URL handling

Module author: python-ldap project (see http://www.python-ldap.org/)

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. Compability note: This module has been solely tested on Python 2.x and above.

See Also:

RFC 4516 - The LDAP URL Format

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`**

1.11.1 LDAPUrl Objects

A `LDAPUrl` object represents a complete LDAP URL.

All class methods:

Class attributes:

Instance attributes:

1.11.2 LDAPUrlExtension Objects

A `LDAPUrlExtension` object represents a single LDAP URL extension.

All class methods:

Class attributes:

Instance attributes:

1.11.3 Example

Important security advice: For security reasons you shouldn't 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.

```
>>> import ldapurl
>>> ldap_url = ldapurl.LDAPUrl('ldap://localhost:1389/dc=stroeder,dc=com?cn,mail
>>> # 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.

```
>>> import ldapurl
>>> ldap_url = ldapurl.LDAPUrl(hostport='localhost:1389',dn='dc=stroeder,dc=com'
>>> ldap_url.unparse()
'ldap://localhost:1389/dc=stroeder,dc=com?cn,mail?base?(objectclass=*)?bindname=
```

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

l

INDEX

P

R

S

T

U

W