LDAP programming with Python

Release 2.0.3

David Leonard, Michael Ströder

October 7, 2004

python-ldap project at Sourceforge
E-mail: python-ldap-dev@lists.sourceforge.net

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.

CONTENTS

CHAPTER

ONE

Python-ldap package

1.1 `ldap` — LDAP library interface module

This module provides access to the LDAP (Lightweight Directory Access Protocol) C API implented in OpenLDAP 2. 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 2.0.3. Source and binaries are available from http://python-ldap.sourceforge.net/.

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 1.1.4) 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 1.1.4) 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.

`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.

`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 various constants.

General

PORT

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

Options

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

OPT_API_FEATURE_INFO

OPT_API_INFO

OPT_CLIENT_CONTROLS

OPT_DEBUG_LEVEL

OPT_DEREF

OPT_ERROR_STRING

OPT_HOST_NAME

OPT_MATCHED_DN

OPT_NETWORK_TIMEOUT

OPT_PRIVATE_EXTENSION_BASE

OPT_PROTOCOL_VERSION

OPT_REFERRALS

OPT_REFHOPLIMIT

OPT_RESTART

OPT_SERVER_CONTROLS

OPT_SIZELIMIT

OPT_SUCCESS

OPT_TIMELIMIT

OPT_TIMEOUT

OPT_URI

OPT_X_SASL_AUTHCID

OPT_X_SASL_AUTHZID

OPT_X_SASL_MECH

OPT_X_SASL_REALM

OPT_X_SASL_SECPROPS

OPT_X_SASL_SSF

OPT_X_SASL_SSF_EXTERNAL

OPT_X_SASL_SSF_MAX

OPT_X_SASL_SSF_MIN

OPT_X_TLS

OPT_X_TLS_ALLOW

OPT_X_TLS_CACERTDIR

OPT_X_TLS_CACERTFILE

OPT_X_TLS_CERTFILE

OPT_X_TLS_CIPHER_SUITE

OPT_X_TLS_CTX

OPT_X_TLS_DEMAND

OPT_X_TLS_HARD

OPT_X_TLS_KEYFILE

OPT_X_TLS_NEVER

OPT_X_TLS_RANDOM_FILE

OPT_X_TLS_REQUIRE_CERT

OPT_X_TLS_TRY

1.1.3 Exceptions

The module defines the following exceptions:

exception `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 `ADMINLIMIT_EXCEEDED`

exception `AFFECTS_MULTIPLE_DSAS`

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. E.g. the *dn* specified with `add()` already exists in the DIT.

exception

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

exception `BUSY`
: The DSA is busy.

exception `CLIENT_LOOP`

exception `COMPARE_FALSE`
: A compare operation returned false. (This exception should never be seen because `compare()` returns a boolean result.)

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

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

exception `CONNECT_ERROR`

exception `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 `CONTROL_NOT_FOUND`

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 methodsearch() (e.g. unbalanced parentheses).

exception `INAPPROPRIATE_AUTH`
: Inappropriate authentication was specified (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()` or `simple_bind()`. (e.g., the wrong password).

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

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

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 `MORE_RESULTS_TO_RETURN`

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

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

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 `NOT_SUPPORTED`

exception `NO_MEMORY`

exception `NO_OBJECT_CLASS_MODS`
Object class modifications are not allowed.

exception `NO_RESULTS_RETURNED`

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 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 `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 is raised if a referral is received when using LDAPv2. (This exception should never be seen with LDAPv3.)

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

exception `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 `SASL_BIND_IN_PROGRESS`

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

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 LDAPObject class

Instances of ldap.LDAPObject are returned by `open()` and `initialize()`. 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.

LDAPObject instances, have the following methods:

`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*)
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 l`ldap.modlist` for generating the modlist.

`bind`(*who, cred, method*)
`bind_s`(*who, cred, method*)
`simple_bind`([*who=''* [*, cred=''*]])
`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`.

`sasl_interactive_bind_s`(*who, auth*)
This call is used to bind to the directory with a SASL bind request.

`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.

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

`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.

`passwd`(*user,oldpw,newpw*)
`passwd_s`(*user,oldpw,newpw*)
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.

The asynchronous version returns the initiated message id.

See Also:

RFC 3062, "*LDAP Password Modify Extended Operation*"

`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` 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 `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.

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.

search(*base, scope* [,*filterstr*=*'(objectClass=*)'* [, *attrlist*=*None* [, *attrsonly*=*0*]]])
search_s(*base, scope* [,*filterstr*=*'(objectClass=*)'* [, *attrlist*=*None* [, *attrsonly*=*0*]]])
search_st(*base, scope* [,*filterstr*=*'(objectClass=*)'* [, *attrlist*=*None* [, *attrsonly*=*0* [, *timeout*=*-1*]]]])
search_ext(*base, scope* [,*filterstr*=*'(objectClass=*)'* [, *attrlist*=*None* [, *attrsonly*=*0* [, *serverctrls*=*None* [, *clientctrls*=*None* [, *timeout*=*-1* [, *sizelimit*=*0*]]]]]]])
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 2254, "*The String Representation of LDAP 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 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.

serverctrls not implemented yet.

clientctrls not implemented yet.

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 `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.

`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

`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.

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*.

`manage_dsa_it``(`*enable,* [*, critical=*`0`]`)`

Enables or disables manageDSAit mode (see draft-zeilenga-ldap-namedref) according to the specified integer flag *enable*. The integer flag *critical* specifies if the use of this extended control is marked critical.

Note This method is somewhat immature and might vanish in future versions if full support for extended controls will be implemented. You have been warned!

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 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.

`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.

`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.

`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()`.

`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*.

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

1.1.5 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=*")
```

1.2 `ldap.async` — Framework for stream-processing of large search results

1.2.1 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
  )
)
```

1.3 `ldap.filter` — LDAP filter handling

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

`escape_filter_chars`(*assertion_value*)

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.

See Also:

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

`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.

1.4 `ldap.modlist` — Generate modify lists

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

`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.

`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.

1.5 `ldap.schema` — Processing LDAPv3 sub schema sub entry

1.5.1 Examples for ldap.schema

```
import ldap.schema
```

1.6 `ldif` — LDIF parser and generator

This module parses and generates LDAP data in the format LDIF.

See Also:

RFC 2849, “*The LDAP Data Interchange Format (LDIF) - Technical Specification*”

1.6.1 Example

The following example demonstrates how to parse an LDIF file with `ldif` module.

```
    To do...
```

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':['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

>>>
```

1.7 `ldapurl` — LDAP URL handling

This module parses and generates LDAP URLs.

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

See Also:

RFC 2255, "*The LDAP URL Format*"

1.7.1 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???bindname=cn=
>>> # 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 `ldapurl` module.

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

MODULE INDEX

INDEX