bonsai Documentation

Release 1.3.0

noirello

Sep 27, 2021

Contents

Bonsai is an LDAP module for Python 3, using OpenLDAP's libldap2 library on Linux, and the WinLDAP library on Microsoft Windows to handle communications with LDAP capable directory servers. The module main goal is to give a simple way to use the LDAP protocol as pythonic as it can be.

Note: The module is compatible only with Python 3.6 or newer releases.

CHAPTER 1

Features

- Uses LDAP libraries (OpenLDAP and WinLDAP) written in C for faster processing.
- Simple pythonic design.
- Implements an own dictionary-like object for mapping LDAP entries that makes easier to add and modify them.
- Works with various asynchronous library (like asyncio, gevent).

CHAPTER 2

Contents

2.1 Installing

2.1.1 Using pip

Bonsai can be simply installed with Pip:

```
$ pip install bonsai
```

2.1.2 Install from source on Linux

These notes illustrate how to compile Bonsai on Linux.

Bonsai is a C wrapper to the OpenLDAP libldap2 library. To install it from sources you will need:

- A C compiler (The module is tested with gcc).
- The Python 3 header files. They are usually installed in a package such as **python3-dev**.
- The libldap header files. They are usually installed in a package such as **libldap2-dev**.
- The libsasl header files. They are usually installed in a package such as **libsasl2-dev**.
- The Bonsai source files. You can download it from the project's GitHub site.
- Optionally for additional functions the Kerberos header files. They are usually installed in a package such as **libkrb5-dev** or **heimdal-dev**.

Once you downloaded and unpackaged the Bonsai source files, you can run the following command to compile and install the package:

```
$ python3 setup.py build
$ sudo python3 setup.py install
```

2.1.3 Install from source on Windows

Bonsai uses WinLDAP on Microsoft Windows. To install it from sources you will need a C compiler and the Bonsai source files. After you downloaded and unpackaged the sources, you can run:

```
$ python setup.py build
$ python setup.py install
```

Note: Compiling the package with MinGW is no longer recommended.

2.1.4 Install from source on Mac OS X

Because Mac OS X is shipped with an older version of libldap which lacks of several features that Bonsai relies on, a newer library needs to be installed before compiling the module.

Install *openldap* homebrew-core formula:

```
$ brew install openldap
```

Modify the *setup.cfg* in the root folder to customize the library and headers directory:

```
[build_ext]
include_dirs=/usr/local/opt/openldap/include
library_dirs=/usr/local/opt/openldap/lib
```

and then you can follow the standard build commands:

```
$ python setup.py build
$ python setup.py install
```

Note: More directories can be set for include and library dirs (e.g. path to the Kerberos headers and libraries) by separating the paths with *:* in the *setup.cfg* file.

After installing Bonsai, you can learn the basic usage in the *Quickstart*.

2.2 Quickstart

After we installed the Bonsai module, let’s see the basic functions to communicate with an LDAP server.

2.2.1 Connecting

First we are connecting to the LDAP server at “example.org” using an `LDAPClient` object:

```
>>> from bonsai import LDAPClient
>>> client = LDAPClient("ldap://example.org")
>>> conn = client.connect()
```

If we want to use a secure connection over SSL/TLS we can change the URL to *ldaps://*:

```
>>> client = LDAPClient("ldaps://example.org")
```

Or set the *tls* parameter to True for the LDAPClient:

```
>>> client = LDAPClient("ldap://example.org", True)
>>> conn = client.connect()
```

Note: Use either the *ldaps://* scheme or the tls parameter set to True for secure connection, but it's ill-advise to use both. If both present the client will set the tls attribute to False to avoid connection error.

If we want to use a filesocket connection point the URL to *ldapi://*:

```
>>> client = LDAPClient("ldapi://%2Frun%2Fslapd%2Fldapi")
```

(Please note that in this case the file location has to be URL-encoded.)

Now, we have an anonym bind to the server, so LDAP whoami operation - which helps to get the identity about the authenticated user - will return with the following:

```
>>> conn.whoami()
'anonymous'
```

To connect with a certain user to the server we have to set credentials before connection:

```
>>> client = LDAPClient("ldaps://example.org")
>>> client.set_credentials("SIMPLE", user="cn=test,dc=bonsai,dc=test", password=
↪"secret")
>>> conn = client.connect()
>>> conn.whoami()
'cn=test,dc=bonsai,dc=test'
```

2.2.2 Searching

To execute a simple search in the dictionary we have to use the *LDAPConnection.search()* method. The function's first parameter - the base DN - sets where we would like to start the search in the dictionary tree, the second parameter - the search scope - can have the following values:

- 0 (base): searching only the base DN.
- 1 (one): searching only one tree level under the base DN.
- 2 (sub): searching of all entries at all levels under, including the base DN.

The scope parameter is replaceable with an *LDAPSearchScope* enumeration, for e.g. *LDAPSearchScope.ONE* for one level search.

The third parameter is a standard LDAP filter string.

The result will be a list of LDAPEntry objects or an empty list, if no object is found.

```
>>> conn = client.connect()
>>> conn.search("ou=nerdherd,dc=bonsai,dc=test", bonsai.LDAPSearchScope.ONE,
↪"(objectclass=*)")
[{'dn': <LDAPDN cn=chuck,ou=nerdherd,dc=bonsai,dc=test>, 'sn': ['Bartowski'],
'cn': ['chuck'], 'givenName': ['Chuck'], 'objectClass': ['inetOrgPerson',
'organizationalPerson', 'person', 'top']}, {'dn': <LDAPDN cn=lester,ou=nerdherd,
↪dc=bonsai,dc=test>,
```

(continues on next page)

(continued from previous page)

```
'sn': ['Patel'], cn': ['lester'], 'givenName': ['Laster'],  'objectClass': [
↪'inetOrgPerson',
'organizationalPerson', 'person', 'top']}, {'dn': <LDAPDN cn=jeff,ou=nerdherd,
↪dc=bonsai,dc=test>,
'sn': ['Barnes'], 'cn': ['jeff'], 'givenName': ['Jeff'], 'objectClass': [
↪'inetOrgPerson',
'organizationalPerson', 'person', 'top']}]
>>> conn.search("ou=nerdherd,dc=bonsai,dc=test", 0, "(objectclass=*)")
[{'dn': <LDAPDN ou=nerdherd,dc=bonsai,dc=test>, 'objectClass': ['organizationalUnit',
↪'top'],
'ou': ['nerdherd']}]
```

The other possible parameters are listed on the API page of `LDAPConnection.search()`.

Note: As you can see every key - or LDAP attribute - in the entry has a list for clarity, even if only one value belongs to the attribute. As most of the attributes could have more than one value, it would be confusing, if some of the keys had string value and the others had list.

2.2.3 Add and modify LDAP entry

To add a new entry to our dictionary we need to create an `LDAPEntry` object with a valid new LDAP DN:

```
>>> from bonsai import LDAPEntry
>>> anna = LDAPEntry("cn=anna,ou=nerdherd,dc=bonsai,dc=test")
>>> anna['objectClass'] = ['top', 'inetOrgPerson'] # Must set schemas to get a valid␣
↪LDAP entry.
>>> anna['sn'] = "Wu" # Must set a surname attribute because inetOrgPerson schema␣
↪requires.
>>> anna['mail'] = "anna@nerdherd.com"
>>> anna.dn
<LDAPDN cn=anna,ou=nerdherd,dc=bonsai,dc=test>
>>> anna
{'dn': <LDAPDN cn=anna,ou=nerdherd,dc=bonsai,dc=test>, 'objectClass': ['top',
↪'inetOrgPerson'],
'sn': ['Wu'], 'mail': ['anna@nerdherd.com']}
```

then call `LDAPConnection.add()` to add to the server:

```
>>> conn.add(anna)
True
```

It's important, that we must set the schemas and every other attribute, that the schemas require. If we miss a required attribute, the server will not finish the operation and return with an `bonsai.ObjectClassViolation` error.

To modify an entry we need to have one that is already in the dictionary (got it back after a search or added it by ourselves previously), then we can easily add new attributes or modify already existing ones like we usually do with a Python dict, the only difference is that we need to call `LDAPEntry.modify()` method at the end to save our modifications on the server side.

```
>>> anna['givenName'] = "Anna" # Set new givenName attribute.
>>> anna['cn'].append('wu') # Add new common name attribute without remove the␣
↪already set ones.
>>> del anna['mail'] # Remove all values of the mail attribute.
```

(continues on next page)

(continued from previous page)

```
>>> anna.modify()
True
```

In certain cases, an LDAP entry can have write-only attribute (e.g. password) that cannot be represented in an LDAPEntry or we just want to change the value of an attribute without reading it first. The `LDAPEntry.change_attribute()` method expects an attribute name, the type of the modification (as an integer or an `LDAPModOp` enum) and the values as parameters to change an entry:

```
>>> from bonsai import LDAPEntry, LDAPModOp
>>> anna = LDAPEntry("cn=anna,ou=nerdherd,dc=bonsai,dc=test")
>>> anna.change_attribute("userPassword", LDAPModOp.REPLACE, "newsecret")
>>> anna.modify()
True
```

2.2.4 Delete an LDAP entry

To delete an entry we've got two options: `LDAPConnection.delete()` and `LDAPEntry.delete()`:

```
>>> conn.delete("cn=anna,ou=nerdherd,dc=bonsai,dc=test") # We have to know the DN of␣
↪the entry.
True
>>> # Or we have a loaded LDAPEntry object, then
>>> anna.delete()
True
```

In the second case the entry is removed on the server, but we still have the data on the client-side.

2.2.5 Rename an LDAP entry

To rename an existing entry call the `LDAPEntry.rename()` method with the new DN on an already loaded `LDAPEntry` object:

```
>>> anna.dn
<LDAPDN cn=anna,ou=nerdherd,dc=bonsai,dc=test>
>>> anna.rename("cn=wu,ou=nerdherd,dc=bonsai,dc=test")
True
>>> anna.dn
<LDAPDN cn=wu,ou=nerdherd,dc=bonsai,dc=test>
```

Be aware that if you would like to move the entry into a different subtree of the directory, then the stated subtree needs to already exist.

2.2.6 Close connection

After we finished our work with the directory server we should close the connection:

```
>>> conn.close()
```

The `LDAPConnection` object can be used with a context manager that will implicitly call the `LDAPConnection.close()` method:

```
import bonsai

cli = bonsai.LDAPClient("ldap://localhost")
with cli.connect() as conn:
    res = conn.search("ou=nerdherd,dc=bonsai,dc=test", 1)
    print(res)
    print(conn.whoami())
```

To find out more about the Bonsai module functionality read the *Advanced Usage* and the *API documentation*.

2.3 Advanced Usage

This document covers some of the more advanced features of Bonsai.

2.3.1 Authentication mechanisms

There are several authentication mechanisms, which can be used with Bonsai. The selected mechanism with the necessary credentials can be set with the `LDAPClient.set_credentials()` method.

Simple bind

The simplest way to authenticate with an LDAP server is using the SIMPLE mechanism which requires a bind DN and a password:

```
>>> import bonsai
>>> client = bonsai.LDAPClient()
>>> client.set_credentials("SIMPLE", user="cn=user,dc=bonsai,dc=test", password=
↪"secret")
>>> client.connect()
<bonsai.ldapconnection.LDAPConnection object at 0x7fed62b19828>
```

Warning: Be aware that during the authentication the password is sent to the server in clear text form. It is ill-advised to use simple bind without secure channel (TLS/SSL) in production.

Simple Authentication and Security Layer

The OpenLDAP library uses the Simple Authentication and Security Layer (SASL) (while the WinLDAP uses the similar SSPI) to provide different authentication mechanisms. Of course to use a certain mechanism it has to be supported both the client and the server. To learn which mechanisms are accessible from the server, check the root DSE's *supportedSASLMechanisms* value:

```
>>> client.get_rootDSE()['supportedSASLMechanisms']
['GSS-SPNEGO', 'GSSAPI', 'DIGEST-MD5', 'NTLM']
```

Note: On Unix systems, make sure that the necessary libraries of the certain mechanism are also installed on the client (e.g. *libsasl2-modules-gssapi-mit* or *cyrus-sasl-gssapi* for GSSAPI support).

DIGEST-MD5 and NTLM

The DIGEST-MD5 and the NTLM mechanisms are challenge-response based authentications. Nowadays they are considered as weak security protocols, but still popular ones. An example of using NTLM:

```
>>> client.set_credentials("NTLM", "user", "secret")
>>> client.connect().whoami()
"dn:cn=user,dc=bonsai,dc=test"
```

The credentials consist of a username and a password, just like for the simple authentication. When using DIGEST-MD5 you can also use an authorization ID during the bind to perform operation under the authority of a different identity afterwards, if the necessary rights are granted for you. NTLM does not support this functionality.

```
>>> client.set_credentials("DIGEST-MD5", "user", "secret", authz_id="u:root")
>>> client.connect().whoami()
"dn:cn=admin,dc=bonsai,dc=test"
```

GSSAPI and GSS-SPNEGO

GSSAPI uses Kerberos tickets to authenticate to the server. To use GSSAPI or GSS-SPNEGO the client must be Kerberos-aware, which means the necessary Kerberos tools and libraries have to be installed, and the proper configuration has to be set. (Typically, the configuration is in the */etc/krb5.conf* on a Unix system). GSS-SPNEGO mechanism can negotiate a common authentication method between server and client.

Basically, to start a GSSAPI authentication a ticket granting ticket (TGT) needs to be already acquired by the client with the help of the command-line *kinit* tool:

```
[noirello@bonsai.test ~]$ kinit admin@BONSAI.TEST
Password for admin@BONSAI.TEST:
```

The acquired TGT can be listed with *klist*:

```
[noirello@bonsai.test ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: admin@BONSAI.TEST

Valid starting     Expires            Service principal
22/02/16 22:06:14  23/02/16 08:06:14  krbtgt/BONSAI.TEST@BONSAI.TEST
    renew until 23/02/16 22:06:12
```

After successfully acquire a TGT, the module can used it for authenticating:

```
>>> import bonsai
>>> client = bonsai.LDAPClient("ldap://bonsai.test")
>>> client.set_credentials("GSSAPI")
>>> client.connect().whoami()
'dn:cn=admin,dc=bonsai,dc=test'
```

In normal case the passed credentials with the exception of the authorization ID are irrelevant – at least on a Unix system, the underlying SASL library figures it out on its own. The module's client can only interfere with the authorization ID:

```
>>> client.set_credentials("GSSAPI", authz_id="u:chuck")
>>> client.connect().whoami()
'dn:cn=chuck,ou=nerdherd,dc=bonsai,dc=test'
```

But on a Windows system (by default) or if Bonsai is built with the optional Kerberos headers, then it is possible to requesting a TGT with the module's client if username, password and realm name are all provided:

```
>>> client = bonsai.LDAPClient("ldap://bonsai.test")
>>> client.set_credentials("GSSAPI", "admin", "secret", "BONSAI.TEST")
>>> client.connect().whoami()
'dn:cn=admin,dc=bonsai,dc=test'
```

It is also possible to use Kerberos keytabs when the module is built with Kerberos support:

```
>>> client.set_credentials("GSSAPI", user="chuck", realm="BONSAI.TEST", keytab="./
↪user.keytab")
>>> client.connect().whoami()
'dn:cn=chuck,ou=nerdherd,dc=bonsai,dc=test'
```

Please note that the Kerberos realm names are typically uppercase with few exceptions.

Note: Automatic TGT requesting only accessible on Unix systems if the optional Kerberos headers are provided during the module's build.

EXTERNAL

With EXTERNAL mechanism TLS certifications are used to authenticate the user. In certain cases (e.g the remote server is an OpenLDAP directory) the EXTERNAL option is presented as an available SASL mechanism only when the client have built up a TLS connection with the server and already set a client cert.

```
>>> client = bonsai.LDAPClient("ldap://bonsai.test", tls=True)
>>> client.set_ca_cert_dir('/etc/openldap/certs')
>>> client.set_ca_cert("RootCACert")
>>> client.set_client_cert("BonsaiTestUser")
>>> client.set_client_key("./key.txt")
>>> client.get_rootDSE()['supportedSASLMechanisms']
['GSS-SPNEGO', 'GSSAPI', 'DIGEST-MD5', 'EXTERNAL', 'NTLM']
>>> client.set_credentials("EXTERNAL")
>>> client.connect()
<bonsai.ldapconnection.LDAPConnection object at 0x7f006ad3d888>
```

For EXTERNAL mechanism only the authorization ID is used in as credential information.

```
>>> client.set_credentials("EXTERNAL", authz_id=u:chuck")
>>> client.connect()
>>> client.connect().whoami()
'dn:cn=chuck,ou=nerdherd,dc=bonsai,dc=test'
```

The proper way of setting the certifications is depend on the TLS implementation that the LDAP library uses. Please for more information see *TLS settings*.

2.3.2 TLS settings

There are two practices to use secure connection:

- Either use the *ldaps://* scheme in the LDAP URL, then the client will use the LDAP over SSL protocol (similar to HTTPS).

- Or set the tls parameter of the `LDAPClient` to True, which will instruct the client to preform a *StartTLS* operation after connected to the LDAP server.

Both practices rely on well-set credentials with the TLS related methods – `LDAPClient.set_ca_cert_dir()`, `LDAPClient.set_ca_cert()`, `LDAPClient.set_client_cert()` and `LDAPClient.set_client_key()`.

These are expecting different inputs depending on which TLS library is used by the LDAP library. To find out which TLS library is used call `bonsai.get_tls_impl_name()`.

GnuTLS and OpenSSL

For GnuTLS and OpenSSL the `LDAPClient.set_ca_cert()` and `LDAPClient.set_client_cert()` are expecting file paths that link to certification files in PEM-format.

The `LDAPClient.set_ca_cert_dir()` works only for OpenSSL if the content of provided directory is symbolic links of certifications that are generated by the *c_rehash* utility.

Mozilla NSS

When using Mozilla NSS the input of `LDAPClient.set_ca_cert_dir()` is the path of the directory containing the NSS certificate database (that is created with the *certutil* command).

The `LDAPClient.set_ca_cert()` and `LDAPClient.set_client_cert()` can be used to select the certificate with their names in the certificate database.

If the client certificate is password protected, then the input of `LDAPClient.set_client_key()` should be a path to the file that contains the password in clear text format.

Microsoft Schannel

Unfortunately, none of the listed TLS modules are effective on Microsoft Windows. The WinLDAP library automatically searches for the corresponding certificates in the cert store. All of the necessary certificates have to be loaded manually before the client tries to use them.

2.3.3 LDAP controls

Several LDAP controls can be used to extend and improve the basic LDAP operations. Bonsai is supporting the following controls. Always check (the root DSE's *supportedControls*) that the server also supports the selected control.

Server side sort

Using the server side sort control the result of the search is ordered based on the selected attributes. To invoke the control simply set the *sort_order* parameter of the `LDAPConnection.search()` method:

```
>>> conn = client.connect()
>>> conn.search("ou=nerdherd,dc=bonsai,dc=test", 2, sort_order=["-cn", "gn"])
```

Attributes that start with - are used for descending order.

Warning: Even if the server side sort control is supported by the server there is no guarantee that the results will be sorted for multiple attributes.

Note: The OID of server side sort control is: 1.2.840.113556.1.4.473.

Paged search result

Paged search can be used to reduce large search result into smaller pages. Page result can be used with the `LDAPConnection.paged_search()` method and the size of the page can be set with the *page_size* parameter:

```
>>> conn = client.connect()
>>> conn.paged_search("ou=nerdherd,dc=bonsai,dc=test", 2, page_size=3)
<_bonsai.ldapsearchiter object at 0x7f006ad455d0>
```

Please note that the return value of `LDAPConnection.paged_search()` is an `ldapsearchiter`. This object can be iterated over the entries of the page. By default the next page of results is acquired automatically during the iteration. This behaviour can be changed by setting the `LDAPClient.auto_page_acquire` to *False* and using the `ldapsearchiter.acquire_next_page()` method which explicitly initiates a new search request to get the next page.

Warning: Avoid using server-side referral chasing with paged search. It's likely to fail with invalid cookie error.

Note: The OID of paged search control is: 1.2.840.113556.1.4.319.

Virtual list view

Virtual list view (VLV) is also for reducing large search result, but with a more specific manner. Virtual list view mimics the scrolling view of an application: it can select a target entry of a large list (ordered search result) with an offset or an attribute value and receiving only a given number of entries before and after it as a partial result of the entire search.

The `LDAPConnection.virtual_list_search()` method's *offset* or *attrvalue* can be used to select the target, the *before_count* and *after_count* for specifying the number of entries before and after the target.

Also need to set the *est_list_count* parameter: the estimated size of the entire list by the client. The server will adjust the position of the target entry based on the real list size, estimated size and the offset.

Virtual list view control cannot be used without a server side sort control thus a sort order always has to be set.

```
>>> conn.virtual_list_search("ou=nerdherd,dc=bonsai,dc=test", 2, attrlist=['cn',
↪'uidNumber'], sort_order=['-uidNumber'], offset=4, before_count=1, after_count=1,␣
↪est_list_count=6)
([{'dn': <LDAPDN cn=sam,ou=nerdherd,dc=bonsai,dc=test>, 'cn': ['sam'], 'uidNumber':␣
↪[4]},
{'dn': <LDAPDN cn=skip,ou=nerdherd,dc=bonsai,dc=test>, 'cn': ['skip'], 'uidNumber':␣
↪[3]},
{'dn': <LDAPDN cn=jeff,ou=nerdherd,dc=bonsai,dc=test>, 'cn': ['jeff'], 'uidNumber':␣
↪[2]}],
{'oid': '2.16.840.1.113730.3.4.10', 'target_position': 4, 'list_count': 7})
```

The return value of the search is a tuple of a list and a dictionary. The dictionary contains the VLV server response: the target position and the real list size.

Note: The OID of virtual list view control is: 2.16.840.1.113730.3.4.9.

Password policy

Password policy defines a set of rules about accounts and modification of passwords. It allows for the system administrator to set expiration time for passwords and a maximal number of failed login attempts before the account become locked. Is also specifies rules about the quality of password.

Enabling the password policy control with *LDAPClient.set_password_policy()* method, the client can receive additional information during connecting to a server or modifying a user's password. Setting this control will change the return value of *LDAPClient.connect()* and *LDAPConnection.open()* to a tuple of *LDAPConnection* and a dictionary that contains the remaining seconds until the password's expiration and the remaining grace logins. The client can also receive new exceptions related to password modifications.

```
>>> import bonsai
>>> client = bonsai.LDAPClient()
>>> client.set_credentials("SIMPLE", "cn=user,dc=bonsai,dc=test", "secret")
>>> client.set_password_policy(True)
>>> conn, ctrl = client.connect()
>>> conn
<bonsai.ldapconnection.LDAPConnection object at 0x7fa552ab4e28>
>>> ctrl
{'grace': 1, 'expire': 3612, 'oid': '1.3.6.1.4.1.42.2.27.8.5.1'})
```

If the server does not support password policy control or the given credentials does not have policies (like anonymous or administrator user) the second item in the tuple will be *None*.

Note: Because the password policy is not standardized, it is not listed by the server among the *supportedControls* even if it is available.

Note: Password policy control cannot be used on MS Windows with WinLDAP. In this case after opening a connection the control dictionary will always be *None*.

Extended DN

Setting LDAP_SERVER_EXTENDED_DN control with *LDAPClient.set_extended_dn()* will extend the standard DN format with the SID and GUID attributes to *<GUID=xxxxxxxx>;<SID=yyyyyyyyy>;distinguishedName* during the LDAP search. The method's parameter can be either 0 which means that the GUID and SID strings will be in a hexadecimal string format or 1 for receiving the extended dn in a standard string format. This control is only supported by Microsoft's Active Directory.

Regardless of setting the control, the *LDAPEntry.dn* still remains a simple *LDAPDN* object without the SID or GUID extensions. The extended DN will be set to the *LDAPEntry.extended_dn* as a string. The extended DN control also affects other LDAP attributes that use distinguished names (e.g. *memberOf* attribute).

```
>>> client = bonsai.LDAPClient()
>>> client.set_extended_dn(1)
>>> result = conn.search("ou=nerdherd,dc=bonsai,dc=test", 1)
>>> result[0].extended_dn
```

(continues on next page)

(continued from previous page)

```
<GUID=899e4e01-e88d-4dea-ba64-119ed386b61c>;<SID=S-1-5-21-101232111302-1767724339-
↪724445543-12345>;cn=chuck,ou=nerdherd,dc=bonsai,dc=test
>>> result[0].dn
<LDAPDN cn=chuck,ou=nerdherd,dc=bonsai,dc=test>
```

Note: The OID of extended DN control is: 1.2.840.113556.1.4.529.

Server tree delete

Server tree delete control allows the client to remove entire subtree with a single request if the user has appropriate permissions to remove every corresponding entry. Setting the *recursive* parameter of `LDAPConnection.delete()` and `LDAPEntry.delete()` to *True* will send the control with the delete request automatically, no further settings are required.

Note: The OID of server tree delete control is: 1.2.840.113556.1.4.805

ManageDsaIT

The ManageDsaIT control can be used to work with LDAP referrals as simple LDAP entries. After setting it with the `LDAPClient.set_managedsait()` method, the referrals can be added removed, and modified just like entries.

```
>>> client = bonsai.LDAPClient()
>>> client.set_managedsait(True)
>>> conn = client.connect()
>>> ref = conn.search("o=admin-ref,ou=nerdherd,dc=bonsai,dc=test", 0)[0]
>>> ref
{'dn': <LDAPDN o=admin-ref,ou=nerdherd,dc=bonsai,dc=test>, 'objectClass': ['referral',
'extensibleObject'], 'o': ['admin-ref']}
>>> type(ref)
<class 'bonsai.ldapentry.LDAPEntry'>
```

Note: The OID of ManageDsaIT control is: 2.16.840.1.113730.3.4.2

SD Flags

The SD Flags control can be used to manage the portion of a Windows security descriptor to retrieve. The flag can be the combination of the following:

- 0x1 - Get the owner identifier of the object.
- 0x2 - Get the primary group identifier.
- 0x4 . Get the discretionary access control list (DACL) of the object.
- 0x8 - Get the system access control list (SACL) of the object.

Use `LDAPClient.set_sd_flags()` method to set the value of the flag.

Note: The OID of SD_FLAGS control is: 1.2.840.113556.1.4.801

2.3.4 Using connection pools

When your application requires to use multiple open LDAP connections, Bonsai provides you connection pools to help you creating and accessing them. This way you can acquire an opened connection, do some operations and put it back into the pool for other threads/tasks to use.

```python
import bonsai
import threading
from bonsai.pool import ThreadedConnectionPool

def work(pool):
    with pool.spawn() as conn:
        print(conn.whoami())
        # Some other operations...

client = bonsai.LDAPClient()
pool = ThreadedConnectionPool(client, minconn=5, maxconn=10)
thr = threading.Thread(target=work, args=(pool,))
thr.start()
conn = pool.get()
res = conn.search()
# After finishing up...
pool.put(conn)
```

2.3.5 Reading and writing LDIF files

Bonsai has a limited support to read and write LDIF files. LDIF (LDAP Data Interchange Format) is a plain text file format for representing LDAP changes and updates. It can be used to exchange data between directory servers.

To read an LDIF file, simply open the file in read-mode, pass it to the *`LDIFReader`*, then the reader object can be used as an iterator to get the entries from the LDIF file.

```python
from bonsai import LDIFReader

with open("users.ldif", "r") as data:
    reader = LDIFReader(data)
    for ent in reader:
        print(ent)
```

Writing LDIF files is similar. The *`LDIFWriter`* needs an open file-object in write-mode, and the *`LDIFWriter.write_entry()`* expects an *`LDAPEntry`* object whose attributes will be serialised. It also possible to serialise the changes of an entry with *`LDIFWriter.write_changes()`*.

```python
from bonsai import LDAPClient
from bonsai import LDIFWriter

client = LDAPClient("ldap://bonsai.test")
with client.connect() as conn:
    res = conn.search("cn=jeff,ou=nerdherd,dc=bonsai,dc=test", 0)
    with open("user.ldif", "w") as data:
        writer = LDIFWriter(data)
```

(continues on next page)

(continued from previous page)

```
        writer.write_entry(res[0])
    # Make some changes on the entry.
    res[0]["mail"].append("jeff_secondary@mail.test")
    res[0]["homeDirectory"] = "/opt/jeff"
    with open("changes.ldif", "w") as data:
        writer = LDIFWriter(data)
        writer.write_changes(res[0])
```

Note: As mentioned above `LDIFReader` and `LDIFWriter` have their limitations. They can handle basic attribute changes (adding, modifying and removing), serialising attributes, but they're not capable to cope with deleting and renaming entries, or processing LDAP controls that are presented in the LDIF file.

2.3.6 Asynchronous operations

Asynchronous operations are first-class citizens in the underlying C API that Bonsai is built on. That makes relatively easy to integrate the module with popular Python async libraries. Bonsai is shipped with support to some: asyncio, gevent, and Tornado.

Using async out-of-the-box

To start asynchronous operations set the `LDAPClient.connect()` method's *is_async* parameter to True. By default the returned connection object can be used with Python's *asyncio* library. For further details about how to use *asyncio* see the official documentation.

An example for asynchronous search and modify with *asyncio*:

```
import asyncio
import bonsai

async def do():
    cli = bonsai.LDAPClient("ldap://localhost")
    async with cli.connect(is_async=True) as conn:
        results = await conn.search("ou=nerdherd,dc=bonsai,dc=test", 1)
        for res in results:
            print(res['givenName'][0])
        search = await conn.search("cn=chuck,ou=nerdherd,dc=bonsai,dc=test", 0)
        entry = search[0]
        entry['mail'] = "chuck@nerdherd.com"
        await entry.modify()

loop = asyncio.get_event_loop()
loop.run_until_complete(do())
```

To work with other non-blocking I/O modules the default asynchronous class has to be set to a different one with `LDAPClient.set_async_connection_class()`.

For example changing it to *GeventLDAPConnection* makes it possible to use the module with gevent:

```
import gevent

import bonsai
from bonsai.gevent import GeventLDAPConnection
```

(continues on next page)

(continued from previous page)

```
def do():
    cli = bonsai.LDAPClient("ldap://localhost")
    # Change the default async conn class.
    cli.set_async_connection_class(GeventLDAPConnection)
    with cli.connect(True) as conn:
        results = conn.search("ou=nerdherd,dc=bonsai,dc=test", 1)
        for res in results:
            print(res['givenName'][0])
        search = conn.search("cn=chuck,ou=nerdherd,dc=bonsai,dc=test", 0)
        entry = search[0]
        entry['mail'] = "chuck@nerdherd.com"
        entry.modify()

gevent.joinall([gevent.spawn(do)])
```

Note: Since 1.2.1, to achieve non-blocking socket connection, the *bonsai.set_connect_async(True)* has to be called before connecting the server. But setting it and using TLS can cause errors even on newer Ubuntu releases (18.04+), thus it has been turned off by default on every platform.

Create your own async class

If you would like to use an asynchronous library that is currently not supported by Bonsai, then you have to work a little bit more to make it possible. The following example will help you to achieve that by showing how to create a new async class for Curio. Inspecting the implementations of the supported libraries can also help.

Warning: This class is just for education purposes, the implementation is made after just scraping the surface of Curio. It's far from perfect and not meant to use in production.

The C API's asynchronous functions are designed to return a message ID immediately after calling them, and then polling the state of the executed operations. The *BaseLDAPConnection* class exposes the same functionality of the C API. Therefore it makes possible to start an operation then poll the result with `LDAPConnection.get_result()` periodically in the *_evaluate* method which happens to be called in every other method that evaluates an LDAP operation.

```
from typing import Optional
import bonsai
import curio

from bonsai.ldapconnection import BaseLDAPConnection

# You have to inherit from BaseLDAPConnection.
class CurioLDAPConnection(BaseLDAPConnection):
    def __init__(self, client: "LDAPClient"):
        super().__init__(client, is_async=True)

    async def _evaluate(self, msg_id: int, timeout: Optional[float] = None):
        while True:
            res = self.get_result(msg_id)
            if res is not None:
```

(continues on next page)

(continued from previous page)

```
                return res
            await curio.sleep(1)
```

The constant polling can be avoided with voluntarily sleep, but it's more efficient to register to an I/O event that will notify when the data is available. The *LDAPConnection.fileno()* method returns the socket's file descriptor that can be used with the OS's default I/O monitoring function (e.g select or epoll) for this purpose. In Curio you can wait until a socket becomes writable with *curio.traps._write_wait*:

```
async def _evaluate(self, msg_id: int, timeout: Optional[float] = None):
    while True:
        await curio.traps._write_wait(self.fileno())
        res = self.get_result(msg_id)
        if res is not None:
            return res
```

The following code is a simple litmus test for proving that the created class plays nice with other coroutines:

```
async def countdown(n):
    while n > 0:
        print(f"T-minus {n}")
        await curio.sleep(1)
        n -= 1

async def search():
    cli = bonsai.LDAPClient()
    cli.set_async_connection_class(CurioLDAPConnection)
    conn = await cli.connect(is_async=True)
    res = await conn.search("ou=nerdherd,dc=bonsai,dc=test", 1)
    for ent in res:
        print(ent.dn)

async def tasks():
    tsk1 = await curio.spawn(countdown, 20)
    tsk2 = await curio.spawn(search)
    await tsk1.join()
    await tsk2.join()

if __name__ == "__main__":
    curio.run(tasks)
```

This example class has the minimal functionalities only but hopefully gives you the basic idea how the asynchronous integration works.

2.3.7 Utilities for Active Directory

Bonsai has some helper classes to work with Windows-specific attributes in Active Directory.

Security descriptor

The Windows security descriptor is a data structure containing the security information associated with a securable object. It's stored in the *ntSecurityDescriptor* attribute as a binary data. To parse it use the *SecurityDescriptor.from_binary()* method:

```
>>> import bonsai
>>> from bonsai.active_directory import SecurityDescriptor
>>> client = LDAPClient("ldap://localhost")
>>> client.set_credentials("SIMPLE", user="cn=chuck,ou=nerdherd,dc=bonsai,dc=test",␣
↪password="secret")
>>> conn = client.connect()
>>> entry = conn.search("cn=chuck,ou=nerdherd,dc=bonsai,dc=test", 0, attrlist=[
↪"ntSecurityDescriptor"])[0]
>>> sec_desc = SecurityDescriptor.from_binary(entry["ntSecurityDescriptor"][0])
>>> print(sec_desc.owner_sid)
<SID: S-1-5-21-3623811015-3361044348-30300820-1013>
>>> print(sec_desc.dacl)
<bonsai.active_directory.acl.ACL object at 0x7f388bc4d6a0>
```

UserAccountControl

The UserAccountControl attribute contains a range of flags which define some important basic properties of a user object like whether the account is active or locked, whether the option of password change at the next logon is enabled, etc. The *`UserAccountControl`* helps to parse this information:

```
>>> from bonsai.active_directory import UserAccountControl
>>> uac = UserAccountControl(entry['userAccountControl'][0])
>>> uac
<bonsai.active_directory.UserAccountControl object at 0x00FC38D0>
>>> uac.properties
{'script': False, 'accountdisable': False, 'homedir_required': False, 'lockout':␣
↪False, 'passwd_notreqd': False, 'passwd_cant_change': False, 'encrypted_text_pwd_
↪allowed': False, 'temp_duplicate_account': False, 'normal_account': True,
↪'interdomain_trust_account': False, 'workstation_trust_account': False, 'server_
↪trust_account': False, 'dont_expire_password': True, 'mns_logon_account': False,
↪'smartcard_required': False, 'trusted_for_delegation': False, 'not_delegated':␣
↪False, 'use_des_key_only': False, 'dont_req_preauth': False, 'password_expired':␣
↪False, 'trusted_to_auth_for_delegation': False, 'partial_secrets_account': False}
```

2.4 API documentation

2.4.1 bonsai

`LDAPClient`

class `bonsai.`**`LDAPClient`**(*url*, *tls=False*)

A class for configuring the connection to the directory server.

Parameters

- **`url`** (*`str`* | *`LDAPURL`*) – an LDAP URL.
- **`tls`** (*`bool`*) – Set *True* to use TLS connection.

Raises **`TypeError`** – if the *url* parameter is not string or not a valid LDAP URL.

`LDAPClient.`**`connect`**(*is_async=False*, *timeout=None*, ***kwargs*)

Open a connection to the LDAP server.

Parameters

- **is_async** (*bool*) – Set *True* to use asynchronous connection.
- **timeout** (*float*) – time limit in seconds for the operation.
- ****kwargs** – additional keyword arguments that are passed to the async connection object (e.g. an eventloop object as *loop* parameter).

Returns an LDAP connection.

Return type *LDAPConnection*

LDAPClient.**get_rootDSE**()

Returns the server's root DSE entry. The root DSE may contain information about the vendor, the naming contexts, the request controls the server supports, the supported SASL mechanisms, features, schema location, and other information.

Returns the root DSE entry.

Return type *LDAPEntry*

An example of getting the root DSE:

```
>>> client = bonsai.LDAPClient()
>>> client.get_rootDSE()
{'namingContexts': ['dc=bonsai,dc=test'], 'supportedControl': ['2.16.840.1.113730.
↪3.4.18',
'2.16.840.1.113730.3.4.2', '1.3.6.1.4.1.4203.1.10.1', '1.2.840.113556.1.4.319',
'1.2.826.0.1.3344810.2.3', '1.3.6.1.1.13.2', '1.3.6.1.1.13.1', '1.3.6.1.1.12'],
'supportedLDAPVersion': ['3'], 'supportedExtension': ['1.3.6.1.4.1.1466.20037',
'1.3.6.1.4.1.4203.1.11.1', '1.3.6.1.4.1.4203.1.11.3', '1.3.6.1.1.8'],
'supportedSASLMechanisms': ['DIGEST-MD5', 'NTLM', 'CRAM-MD5']}
```

LDAPClient.**set_async_connection_class**(*conn*)

Set the LDAP connection class for asynchronous connection. The default connection class is *bonsai.asyncio.AIOLDAPConnection* that uses the asyncio event loop.

Parameters **conn** (*BaseLDAPConnection*) – the new asynchronous connection class that is a subclass of LDAPConnection.

Raises **TypeError** – if *conn* parameter is not a subclass of BaseLDAPConnection.

An example to change the default async connection class to a Gevent-based one:

```
>>> import bonsai
>>> from bonsai.gevent import GeventLDAPConnection
>>> client = bonsai.LDAPClient()
>>> client.set_async_connection_class(GeventLDAPConnection)
>>> client.connect(True)
<bonsai.gevent.geventconnection.GeventLDAPConnection object at 0x7f9b1789c6d8>
```

LDAPClient.**set_auto_page_acquire**(*val*)

Turn on or off the automatic page acquiring during a paged LDAP search. By turning automatic page acquiring on, it is unnecessary to call *ldapsearchiter.acquire_next_page()*. It will be implicitly called during iteration.

Parameters **val** (*bool*) – enabling/disabling auto page acquiring.

Raises **TypeError** – If the parameter is not a bool type.

LDAPClient.**set_ca_cert**(*name*)

Set the name of CA certificate. If the underlying libldap library uses the Mozilla NSS as TLS library the *name* should be the same one in the cert/key database (that specified with *LDAPClient.set_ca_cert_dir()*), otherwise it can be the name of the CA cert file.

Note: This method has no effect on MS Windows, because WinLDAP searches for the corresponding CA certificate in the cert store. This means that the necessary certificates have to be installed manually in to the cert store.

Parameters **`name`** (*`str`*) – the name of the CA cert.

Raises **`TypeError`** – if *name* parameter is not a string or not None.

`LDAPClient.`**`set_ca_cert_dir`**`(`*path*`)`

Set the directory of the CA cert. If the underlying libldap library uses the Mozilla NSS as TLS library the *path* should be the path to the existing cert/key database, otherwise it can be the path of the CA cert file.

Note: This method has no effect on MS Windows, because WinLDAP searches for the corresponding CA certificate in the cert store. This means that the necessary certifications have to be installed manually in to the cert store.

Parameters **`path`** (*`str`*) – the path to the CA directory.

Raises **`TypeError`** – if *path* parameter is not a string or not None.

`LDAPClient.`**`set_cert_policy`**`(`*policy*`)`

Set policy about server certification.

Parameters **`policy`** (*`str`*) – the cert policy could be one of the following strings:

- *try* or *demand*: the server cert will be verified, and if it fail, then the *`LDAPClient.connect()`* will raise an error.
- *never* or *allow*: the server cert will be used without any verification.

Raises

- **`TypeError`** – if the *policy* parameter is not a string.
- **`ValueError`** – if the *policy* not one of the four above.

Warning: Set off the cert verification is dangerous. Without verification there is a chance of man-in-the-middle attack.

`LDAPClient.`**`set_client_cert`**`(`*name*`)`

Set the name of client certificate. If the underlying libldap library uses the Mozilla NSS as TLS library the *name* should be the same one in the cert/key database (that specified with *`LDAPClient.set_ca_cert_dir()`*), otherwise it can be the name of the client certificate file.

Note: This method has no effect on MS Windows, because WinLDAP searches for the corresponding client certificate based on the servert's CA cert in the cert store. This means that the necessary certificates have to be installed manually in to the cert store.

Parameters **`name`** (*`str`*) – the name of the client cert.

Raises **`TypeError`** – if *name* parameter is not a string or not None.

LDAPClient.**set_client_key**(*name*)

Set the file that contains the private key that matches the certificate of the client that specified with *LDAPClient.set_client_cert()*.

Note: This method has no effect on MS Windows, because WinLDAP searches for the corresponding client certificate based on the servert's CA cert in the cert store. This means that the necessary certificates have to be installed manually in to the cert store.

Parameters **name** (*str*) – the name of the CA cert.

Raises **TypeError** – if *name* parameter is not a string or not None.

LDAPClient.**set_credentials**(*mechanism*, *user=None*, *password=None*, *realm=None*, *authz_id=None*, *keytab=None*)

Set binding mechanism and credential information. All parameters are optional except the *mechanism*. Different mechanism applies different credential information and ignores the rest. For example:

- *SIMPLE* uses the *user* (as bind DN) and *password*.
- *EXTERNAL* only uses the *authz_id* as authorization ID.

For other use-cases see this section about *authentication mechanisms*.

Parameters

- **mechanism** (*str*) – the name of the binding mechanism.
- **user** (*str*) – the identification of the binding user.
- **password** (*str*) – the password of the user.
- **realm** (*str*) – the (Kerberos) realm of the user.
- **authz_id** (*str*) – the authorization ID for the user.
- **keytab** (*str*) – path to a Kerberos keytab for authentication.

Raises **TypeError** – if mechanism is not string, or any of the other parameters are not string or None, or both of the password and keytab parameters are set.

```
>>> from bonsai import LDAPClient
>>> client = LDAPClient()
>>> client.set_credentials("SIMPLE", user="cn=user,dc=bonsai,dc=test", password=
↪"secret")
>>> client.connect()
<bonsai.LDAPConnection object at 0x7fadf8976440>
```

LDAPClient.**set_extended_dn**(*extdn_format*)

Set the format of extended distinguished name for LDAP_SERVER_EXTENDED_DN_OID control which extends the entries' distingushed name with GUID and SID attributes. If the server supports the control, the LDAPEntry objects' *extended_dn* attribute will be set (as a string) and the *dn* attribute will be kept in the simple format.

Setting 0 specifies that the GUID and SID values be returned in hexadecimal string format, while setting 1 will return the GUID and SID values in standard string format. Passing *None* will remove the control in a format of *<GUID=xxxx>;<SID=yyyy>;distinguishedName*.

Parameters **extdn_format** (*int*) – the format of the extended dn. It can be 0, 1 or *None*.

Raises

- **TypeError** – if the parameter is not int or None.
- **ValueError** – if the parameter is not 0, 1 or None.

An example:

```
>>> client = bonsai.LDAPClient()
>>> client.set_extended_dn(1)
>>> result = conn.search("ou=nerdherd,dc=bonsai,dc=test", 1)
>>> result[0].extended_dn
<GUID=899e4e01-e88d-4dea-ba64-119ed386b61c>;<SID=S-1-5-21-101232111302-1767724339-
↪724445543-12345>;cn=chuck,ou=nerdherd,dc=bonsai,dc=test
>>> result[0].dn
<LDAPDN cn=chuck,ou=nerdherd,dc=bonsai,dc=test>
```

Note: If the extended dn control is not supported the LDAPEntry's extended_dn attribute will be None. The LDAP_SERVER_EXTENDED_DN_OID is defined as '1.2.840.113556.1.4.529'.

LDAPClient.**set_ignore_referrals**(*val*)

Turn on or off ignoring LDAP referrals in search result. When enabled (default), then possible LDAP referrals won't be represented in the search result. When disabled and *LDAPClient.server_chase_referrals* is also disabled, than search result can contain *LDAPReference* objects along with *LDAPEntry* objects.

Parameters **val** (*bool*) – enabling/disabling LDAP referrals in search result.

Raises **TypeError** – If the parameter is not a bool type.

LDAPClient.**set_managedsait**(*val*)

Set ManageDsaIT control for LDAP operations. With ManageDsaIT an LDAP referral can be searched, added and modified as a common LDAP entry.

Parameters **val** (*bool*) – enabling/disabling ManageDsaIT control.

Raises **TypeError** – If the parameter is not a bool type.

LDAPClient.**set_password_policy**(*ppolicy*)

Enable password policy control, if it is provided by the directory server. Setting it *True* will change the return value of *LDAPClient.connect()* and *LDAPConnection.open()* to a tuple of *(conn, ctrl)* where the *conn* is an *LDAPConnection*, the *ctrl* is a dict of returned password policy control response that contains the oid, the remaining seconds of password expiration, and the number of remaining grace logins. If the password policy control is not available on the server or not supported by the platform the second item in the returned tuple is *None*, instead of a dictionary.

By enabling the password policy control the server can send additional error messages related to the user's account and password during conneting to the server and changing entries.

Parameters **ppolicy** (*bool*) – enabling/disabling password policy control.

Raises **TypeError** – If the parameter is not a bool type.

An example:

```
>>> import bonsai
>>> client = bonsai.LDAPClient()
>>> client.set_credentials("SIMPLE", "cn=user,dc=bonsai,dc=test", "secret")
>>> client.set_password_policy(True)
>>> conn, ctrl = client.connect()
>>> conn
<bonsai.ldapconnection.LDAPConnection object at 0x7fa552ab4e28>
```

(continues on next page)

(continued from previous page)

```
>>> ctrl
{'grace': 1, 'expire': 3612, 'oid': '1.3.6.1.4.1.42.2.27.8.5.1'})
```

Note: Password policy control cannot be used on MS Windows with WinLDAP. In this case after opening a connection the control dictionary will always be *None*.

`LDAPClient.`**`set_raw_attributes`**`(`*raw_list*`)`

By default the values of the LDAPEntry are in string format. The values of the listed LDAP attribute's names in *raw_list* will be kept in bytearray format.

Parameters **`raw_list`** (*`list`*) – a list of LDAP attribute's names. The elements must be string and unique.

Raises

- **`TypeError`** – if any of the list's element is not a string.
- **`ValueError`** – if the item in the lit is not a unique element.

An example:

```
>>> client = bonsai.LDAPClient()
>>> client.set_raw_attributes(["cn", "sn"])
>>> conn = client.connect()
>>> conn.search("cn=jeff,ou=nerdherd,dc=bonsai,dc=test", 0, attrlist=['cn', 'sn',
↪'gn'])
[{'dn': <LDAPDN cn=jeff,ou=nerdherd,dc=bonsai,dc=test>, 'sn': [b'Barnes'], 'cn':␣
↪[b'jeff'],
'givenName': ['Jeff']}]
```

`LDAPClient.`**`set_sd_flags`**`(`*flags*`)`

Set the flags for retreiving specific portion of the Windows security descriptor.

Parameters **`flags`** (*`int | None`*) – the flags for the security descriptor.

Raises **`TypeError`** – if the parameter is not int or None.

`LDAPClient.`**`set_server_chase_referrals`**`(`*val*`)`

Turn on or off chasing LDAP referrals by the server. By turning off server-side referral chasing along with *`LDAPClient.ignore_referrals`*, search result can contain *`LDAPReference`* objects mixed with *`LDAPEntry`* objects.

Parameters **`val`** (*`bool`*) – enabling/disabling LDAP referrals chasing.

Raises **`TypeError`** – If the parameter is not a bool type.

`LDAPClient.`**`set_url`**`(`*url*`)`

Set LDAP url for the client.

Parameters **`url`** (*`LDAPURL | str`*) – the LDAP url.

`LDAPClient.`**`auto_page_acquire`**

The status of automatic page acquiring. *True* by default.

`LDAPClient.`**`ca_cert`**

The name of the CA certificate.

`LDAPClient.`**`ca_cert_dir`**

The path to the CA certificate.

LDAPClient.**cert_policy**
The certification policy.

LDAPClient.**client_cert**
The name of the client certificate.

LDAPClient.**client_key**
The key file to the client's certificate.

LDAPClient.**credentials**
A dict with the credential information. It cannot be set.

LDAPClient.**extended_dn_format**
Format of the extended distinguished name. 0 means hexadecimal string format, 1 standard string format. If it is *None*, then it's not set.

LDAPClient.**ignore_referrals**
The status of ignoring referrals in search results. *True* by default.

LDAPClient.**managedsait**
The status of using ManageDsaIT control.

LDAPClient.**mechanism**
The chosen mechanism for authentication. It cannot be set.

LDAPClient.**password_policy**
The status of using password policy.

LDAPClient.**raw_attributes**
A list of attributes that should be kept in byte format.

LDAPClient.**sd_flags**
The flags for the Windows security descriptor.

LDAPClient.**server_chase_referrals**
The status of chasing referrals by the server. *False* by default.

Changed in version 1.3.0: Default value from *True* to *False*.

LDAPClient.**tls**
A bool about TLS connection is required. It cannot be set.

LDAPClient.**url**
The URL of the directory server.

LDAPConnection

class bonsai.**LDAPConnection**(*client: LDAPClient*)
Handles synchronous connection to an LDAP server.

Parameters **client** (LDAPClient) – a client object.

LDAPConnection.**abandon**(*msg_id*)
Abandon an ongoing asynchronous operation associated with the given message id. Note that there is no guarantee that the LDAP server will be able to honor the request, which means the operation could be performed anyway. Nevertheless, it is a good programming paradigm to abandon unwanted operations (e.g after a timeout is exceeded).

Parameters **msg_id** (*int*) – the ID of an ongoing LDAP operation.

LDAPConnection.**add**(*entry*, *timeout=None*)
Add new entry to the directory server.

Parameters

- **entry** (LDAPEntry) – the new entry.
- **timeout** (*float*) – time limit in seconds for the operation.

Returns True, if the operation is finished.

Return type bool

LDAPConnection.**close**()

Close LDAP connection.

LDAPConnection.**delete**(*dname*, *timeout=None*, *recursive=False*)

Remove entry from the directory server.

Parameters

- **dname** (*str | LDAPDN*) – the string or LDAPDN format of the entry's DN.
- **timeout** (*float*) – time limit in seconds for the operation.
- **recursive** (*bool*) – remove every entry of the given subtree recursively.

Returns True, if the operation is finished.

Return type bool

LDAPConnection.**fileno**()

Return the file descriptor of the underlying socket that is used for the LDAP connection.

Returns The file descriptor.

Return type int

LDAPConnection.**get_result**(*msg_id*, *timeout=None*)

Get the result of an ongoing asynchronous operation associated with the given message id. The method blocks the caller until the given *timeout* parameter is passed or the result is arrived. If the operation is not finished until the timeout, it returns None. If the *timeout* is None, it returns immediately.

Parameters

- **msg_id** (*int*) – the ID of an ongoing LDAP operation.
- **timeout** (*float*) – time limit in seconds for waiting on the result.

Returns the result of the operation.

Return type depending on the type of the operation.

Raises ***bonsai.InvalidMessageID*** – if the message ID is invalid or the associated operation is already finished

LDAPConnection.**open**(*timeout=None*)

Open the LDAP connection.

Parameters **timeout** (*float*) – time limit in seconds for the operation.

Returns The *LDAPConnection* object itself.

Return type *LDAPConnection*.

LDAPConnection.**modify_password**(*user=None*, *new_password=None*, *old_password=None*, *timeout=None*)

Set a new password for the given user.

Parameters

- **user** (*str | LDAPDN*) – the identification of the user. If not set, the owner of the current LDAP session will be associated.
- **new_password** (*str*) – the new password. If not set, the server will generate one and the new password will be returned by this method.
- **old_password** (*str*) – the current password of the user.
- **timeout** (*float*) – time limit in seconds for the operation.

Returns if the *new_password* is not set, then the generated password, None otherwise.

Return type str|None

See also:

RFC about the LDAP Password Modify extended operation RFC3062.

`LDAPConnection.`**`search`**(*base=None*, *scope=None*, *filter_exp=None*, *attrlist=None*, *timeout=None*, *sizelimit=0*, *attrsonly=False*, *sort_order=None*)

Perform a search on the directory server. A base DN and a search scope is always necessary to perform a search, but these values - along with the attribute's list and search filter - can also be set with the `LDAPClient` LDAP URL parameter. The parameters, which are passed to the `LDAPConnection.search()` method will overrule the previously set ones with the LDAP URL.

Setting *sort_order* will invoke server side sorting LDAP control, based on the provided attribute list.

```
>>> from bonsai import LDAPClient
>>> client = LDAPClient("ldap://localhost") # without additional parameters
>>> conn = client.connect()
>>> conn.search("ou=nerdherd,dc=bonsai,dc=test", 1, "(cn=ch*)", ["cn", "sn", "gn
↪"])
[{'dn': <LDAPDN cn=chuck,ou=nerdherd,dc=bonsai,dc=test>, 'sn': ['Bartowski'], 'cn
↪': ['chuck'],
'givenName': ['Chuck']}]
>>> client = LDAPClient("ldap://localhost/ou=nerdherd,dc=bonsai,dc=test?cn,sn,gn?
↪one?(cn=ch*)") # with additional parameters
>>> conn = client.connect()
>>> conn.search()
[{'dn': <LDAPDN cn=chuck,ou=nerdherd,dc=bonsai,dc=test>, 'sn': ['Bartowski'], 'cn
↪': ['chuck'],
'givenName': ['Chuck']}]
>>> conn.search(filter_exp="(cn=j*)")
[{'dn': <LDAPDN cn=jeff,ou=nerdherd,dc=bonsai,dc=test>, 'sn': ['Barnes'], 'cn': [
↪'jeff'],
'givenName': ['Jeff']}]
```

Parameters

- **base** (*str*) – the base DN of the search.
- **scope** (*int*) – the scope of the search. An `LDAPSearchScope` also can be used as value.
- **filter_exp** (*str*) – string to filter the search in LDAP search filter syntax.
- **attrlist** (*list*) – list of attribute's names to receive only those attributes from the directory server.
- **timeout** (*float*) – time limit in seconds for the search.
- **sizelimit** (*int*) – the number of entries to limit the search.

- **attrsonly** (*bool*) – if it's set True, search result will contain only the name of the attributes without their values.
- **sort_order** (*list*) – list of attribute's names to use for server-side ordering, start name with '-' for descending order.

Returns the search result.

Return type list

`LDAPConnection.`**`paged_search`**(*base=None*, *scope=None*, *filter_exp=None*, *attrlist=None*, *timeout=None*, *sizelimit=0*, *attrsonly=False*, *sort_order=None*, *page_size=1*)

Perform a search that returns a paged search result. The number of entries on a page is limited with the *page_size* parameter. The return value is an `ldapsearchiter` which is an iterable object. By default, after returning the last entry on the page it automatically requests the next page from the server until the final page is delivered. This functionality can be disabled by setting the `LDAPClient.auto_page_acquire` to *false*. Then the next page can be acquired manually by calling the `ldapsearchiter.acquire_next_page()` method.

Parameters

- **base** (*str*) – the base DN of the search.
- **scope** (*int*) – the scope of the search. An `LDAPSearchScope` also can be used as value.
- **filter_exp** (*str*) – string to filter the search in LDAP search filter syntax.
- **attrlist** (*list*) – list of attribute's names to receive only those attributes from the directory server.
- **timeout** (*float*) – time limit in seconds for the search.
- **sizelimit** (*int*) – the number of entries to limit the search.
- **attrsonly** (*bool*) – if it's set True, search result will contain only the name of the attributes without their values.
- **sort_order** (*list*) – list of attribute's names to use for server-side ordering, start name with '-' for descending order.
- **page_size** (*int*) – the number of entries on a page.

Returns the search result.

Return type ldapsearchiter

`LDAPConnection.`**`virtual_list_search`**(*base=None*, *scope=None*, *filter_exp=None*, *attrlist=None*, *timeout=None*, *sizelimit=0*, *attrsonly=False*, *sort_order=None*, *offset=1*, *before_count=0*, *after_count=0*, *est_list_count=0*, *attrvalue=None*)

Perform a search using virtual list view control. To perform the search the server side sort control has to be set with *sort_order*. The result set will be shifted to the *offset* or *attrvalue* and contains the specific number of entries after and before, set with *after_count* and *before_count*. The *est_list_count* is an estimation of the entire searched list that helps to the server to position the target entry.

The result of the operation is a tuple of a list and a dictionary. The dictionary contains the VLV server response: the target position and the real list size. Thi list contains the searched entries.

For further details using these controls please see *LDAP controls*.

Parameters

- **base** (*str*) – the base DN of the search.

- **scope** (*int*) – the scope of the search. An *LDAPSearchScope* also can be used as value.
- **filter_exp** (*str*) – string to filter the search in LDAP search filter syntax.
- **attrlist** (*list*) – list of attribute's names to receive only those attributes from the directory server.
- **timeout** (*float*) – time limit in seconds for the search.
- **sizelimit** (*int*) – the number of entries to limit the search.
- **attrsonly** (*bool*) – if it's set True, search result will contain only the name of the attributes without their values.
- **sort_order** (*list*) – list of attribute's names to use for server-side ordering, start name with '-' for descending order.
- **offset** (*int*) – an offset of the search result to select a target entry for virtual list view (VLV).
- **before_count** (*int*) – the number of entries before the target entry for VLV.
- **after_count** (*int*) – the number of entries after the target entry for VLV.
- **est_list_count** (*int*) – the estimated content count of the entire list for VLV.
- **attrvalue** – an attribute value (of the attribute that is used for sorting) for identifying the target entry for VLV.

Returns the search result.

Return type (list, dict)

LDAPConnection.**whoami**(*timeout=None*)

This method can be used to obtain authorization identity.

Parameters **timeout** (*float*) – time limit in seconds for the operation.

Returns the authorization ID.

Return type str

See also:

RFC about the LDAP Who am I extended operation RFC4532.

LDAPConnection.**closed**

A readonly attribute about the connection's state.

LDAPConnection.**is_async**

A readonly attribute to define that the connections is asynchronous.

LDAPDN

Class for representing LDAP distinguished names.

See also:

RFC about **LDAP: String Representation of Distinguished Names** RFC4514.

Example for working with LDAPDN objects.

```
>>> import bonsai
>>> dn = bonsai.LDAPDN("cn=testuser,dc=bonsai,dc=test")
>>> dn
<LDAPDN cn=testuser,dc=bonsai,dc=test>
>>> dn.rdns # Get RDNs in tuple format.
((('cn', 'testuser'),), (('dc', 'bonsai'),), (('dc', 'test'),))
>>> str(dn) # Convert to string.
'cn=testuser,dc=bonsai,dc=test'
>>> dn[1] # Get the second RDN.
'dc=bonsai'
>>> dn[0] # Get the first RDN.
'cn=testuser'
>>> dn[1] = "ou=nerdherd,dc=bonsai" # Change the second RDN.
>>> dn
<LDAPDN cn=testuser,ou=nerdherd,dc=bonsai,dc=test>
>>> other_dn = bonsai.LDAPDN("cn=testuser,ou=nerdherd,dc=bonsai,dc=test")
>>> dn == other_dn
True
>>> dn[1:3] # Get the second and third RDN.
'ou=nerdherd,dc=bonsai'
>>> dn[1:3] = 'ou=buymore,dc=bonsai' # Change them.
>>> dn
<LDAPDN cn=testuser,ou=buymore,dc=bonsai,dc=test>
```

class bonsai.**LDAPDN**(*strdn: str*)

A class for handling valid LDAP distinguished name.

Parameters **strdn** (*str*) – a string representation of LDAP distinguished name.

LDAPDN.**__getitem__**(*idx*)

Return the string format of the relative distinguished names in the LDAPDN.

Parameters **idx** (*int*) – the indices of the RDNs.

Returns the string format of the RDNs.

Return type str

LDAPDN.**__setitem__**(*idx*, *value*)

Set the string format of the relative distinguished names in the LDAPDN.

Parameters

- **idx** (*int*) – the indices of the RDNs.
- **value** (*str*) – the new RDNs.

LDAPDN.**__eq__**(*other*)

Check equality of two LDAPDNs by their string formats or their sanitized string formats.

LDAPDN.**__str__**() → str

Return the full string format of the distinguished name.

LDAPDN.**rdns**

The tuple of relative distinguished name.

LDAPEntry

class bonsai.**LDAPEntry**(*dn*[, *conn*])

LDAPEntry.**change_attribute**(*name*, *optype*, **values*)

Change an attribute of the entry with explicit LDAP modification type by listing the values as parameters. An attribute can be removed entirely if the *optype* is delete and no *values* are passed. This method can be useful for changing write-only attributes (e.g. passwords).

Parameters

- **name** (*str*) – the name of the attribute.
- **optype** (*int*) – the operation type, 0 for adding, 1 for deleting and 2 for replacing. An *LDAPModOp* also can be used as value.
- ***values** – the new value or values of the attribute.

Note: It is possible to create an inconsistent state that the server will reject. To get a clean state use *LDAPEntry.clear_attribute_changes()*.

LDAPEntry.**clear**() → None

Remove all items from the dictionary.

LDAPEntry.**clear_attribute_changes**(*name*)

Clear all added and deleted changes of an attribute.

Parameters **name** (*str*) – the name of the attribute.

LDAPEntry.**delete**(*timeout=None*, *recursive=False*)

Remove LDAP entry from the dictionary server.

Parameters

- **timeout** (*float*) – time limit in seconds for the operation.
- **recursive** (*bool*) – remove every entry of the given subtree recursively.

Returns True, if the operation is finished.

Return type bool

LDAPEntry.**get**(*key*, *default=None*)

Return the value for *key* if *key* is in the LDAPEntry, else *default*. (Same as dict's get method.)

LDAPEntry.**items**(*exclude_dn=False*)

Same functionality as dict.items() but with an optional *exclude_dn* keyword-only argument. When *exclude_dn* is True it returns a generator of the LDAPEntry's key-value pairs without the dn key and value.

Parameters **exclude_dn** (*bool*) – exclude dn key from the list.

Returns sequence of key-value pairs.

Return type dict_items, generator

LDAPEntry.**keys**(*exclude_dn=False*)

Same functionality as dict.keys() but with an optional *exclude_dn* keyword-only argument. When *exclude_dn* is True it returns a generator of the LDAPEntry's keys without the dn key.

Parameters **exclude_dn** (*bool*) – exclude dn key from the list.

Returns sequence of keys.

Return type dict_keys, generator

Note: Be aware when the *exclude_dn* argument of *`LDAPEntry.items()`*, *`LDAPEntry.keys()`* or *`LDAPEntry.values()`* is set to *True* you lose the benefits of dict views and get a generator object that will be sensitive of adding and removing items to the entry.

`LDAPEntry.`**`modify`**(*timeout=None*)

Send entry's modifications to the dictionary server.

Parameters **`timeout`** (*`float`*) – time limit in seconds for the operation.

Returns True, if the operation is finished.

Return type bool

`LDAPEntry.`**`rename`**(*newdn*, *timeout=None*, *delete_old_rdn=True*)

Change the entry's distinguished name.

Parameters

- **`newdn`** (*`str | LDAPDN`*) – the new DN of the entry.
- **`timeout`** (*`float`*) – time limit in seconds for the operation.
- **`delete_old_rdn`** (*`bool`*) – remove old RDN with renaming.

Returns True, if the operation is finished.

Return type bool

`LDAPEntry.`**`update`**(**args*, ***kwds*)

Update the LDAPEntry with the key/value pairs from other, overwriting existing keys. (Same as dict's update method.)

`LDAPEntry.`**`values`**(*exclude_dn=False*)

Same functionality as dict.values() but with an optional *exclude_dn* keyword-only argument. When *exclude_dn* is True it returns a generator of the LDAPEntry's values without the value of the dn key.

Parameters **`exclude_dn`** (*`bool`*) – exclude dn key from the list.

Returns sequence of values.

Return type dict_values, generator

`LDAPEntry.`**`connection`**

The LDAPConnection object of the entry. Needs to be set for any LDAP operations.

`LDAPEntry.`**`dn`**

The distinguished name of the entry.

```
>>> from bonsai import LDAPEntry
>>> anna = LDAPEntry('cn=anna,ou=nerdherd,dc=bonsai,dc=test')
>>> anna.dn
<LDAPDN cn=anna,ou=nerdherd,dc=bonsai,dc=test>
>>> str(anna.dn)
'cn=anna,ou=nerdherd,dc=bonsai,dc=test'
```

`LDAPEntry.`**`extended_dn`**

The extended DN of the entry. It is None, if the extended DN control is not set or not supported. The attribute is read-only.

LDAPModOp

class bonsai.**LDAPModOp**

Enumeration for LDAP modification operations.

LDAPModOp.**ADD = 0**

For adding new values to the attribute.

LDAPModOp.**DELETE = 1**

For deleting existing values from the attribute list.

LDAPModOp.**REPLACE = 2**

For replacing the existing attribute values.

LDAPReference

class bonsai.**LDAPReference**(*client*, *references*)

Object for handling an LDAP reference.

Parameters

- **client** (LDAPClient) – a client object.
- **references** (*list*) – list of valid LDAP URLs (as string or *LDAPURL* objects).

LDAPReference.**client**

The LDAP client.

LDAPReference.**references**

The list of LDAPURLs of the references.

LDAPSearchScope

class bonsai.**LDAPSearchScope**

Enumeration for LDAP search scopes.

LDAPSearchScope.**BASE = 0**

For searching only the base DN.

LDAPSearchScope.**ONELEVEL = 1**

For searching one tree level under the base DN.

LDAPSearchScope.**ONE = 1**

Alias for *LDAPSearchScope.ONELEVEL*.

LDAPSearchScope.**SUBTREE = 2**

For searching the entire subtree, including the base DN.

LDAPSearchScope.**SUB = 2**

Alias for *LDAPSearchScope.SUBTREE*.

LDAPURL

See also:

RFC about **LDAP: Uniform Resource Locator** RFC4516.

class bonsai.**LDAPURL**(*str*)

LDAP URL object for handling LDAP connection informations, such as hostname, port, LDAP bind DN, search attributes, scope (base,sub or one) filter, and extensions. If *strurl* is None, then the default url is *ldap://localhost:389*.

Parameters **strurl** (`str`) – string representation of a valid LDAP URL. Must be started with *ldap://*, *ldaps://* or *ldapi://*.

Raises **ValueError** – if the string parameter is not a valid LDAP URL.

An example of a valid LDAP URL with port number, base DN, list of attributes and search filter:

```
>>> from bonsai import LDAPURL
>>> url = LDAPURL("ldap://localhost:789/ou=nerdherd,dc=bonsai,dc=test?cn,sn,gn?
↪sub?(cn=c*)")
>>> url
<LDAPURL ldap://localhost:789/ou=nerdherd,dc=bonsai,dc=test?cn,sn,gn?sub?(cn=c*)>
>>> url.basedn
<LDAPDN ou=nerdherd,dc=bonsai,dc=test>
>>> url.attributes
['cn', 'sn', 'gn']
```

LDAPURL.**get_address**() → str

Return the full address of the host.

```
>>> import bonsai
>>> url = bonsai.LDAPURL("ldaps://example.com/cn=test,dc=bonsai,dc=test??sub")
>>> url
<LDAPURL ldaps://example.com:636/cn=test,dc=bonsai,dc=test??sub>
>>> url.get_address()
'ldaps://example.com:636'
```

LDAPURL.**__str__**() → str

Returns the full format of LDAP URL.

LDAPURL.**attributes**

The searching attributes.

LDAPURL.**basedn**

The LDAP distinguished name for binding.

LDAPURL.**host**

The hostname.

LDAPURL.**port**

The portnumber.

LDAPURL.**filter_exp**

The searching filter expression.

LDAPURL.**scope**

The searching scope.

LDAPURL.**scope_num**

Return the searching scope number.

LDAPURL.**scheme**

The URL scheme.

LDAPValueList

class bonsai.**LDAPValueList**(*items*)

Modified list that tracks added and deleted values. It also contains only unique elements. The elements are compared to their lower-cased string representations.

A new LDAPValueList can be created optionally from an existing sequence object.

Parameters **items** – a sequence object.

Raises **ValueError** – if *items* has a non-unique element.

LDAPValueList.**__contains__**(*item*)

Return key in self.

LDAPValueList.**__delitem__**(*idx*)

Delete self[key].

LDAPValueList.**__setitem__**(*idx*, *value*)

Set self[key] to value.

LDAPValueList.**__add__**(*other*)

Return self+value.

LDAPValueList.**__iadd__**(*other*)

Implement self+=value.

LDAPValueList.**__mul__**(*value*)

Return self*value.

LDAPValueList.**append**(*item*)

Add a unique item to the end of the LDAPValueList.

Parameters **item** – New item.

Raises **ValueError** – if the *item* is not unique.

LDAPValueList.**extend**(*items*)

Extend the LDAPValueList by appending all the items in the given list. All element in *items* must be unique and also not represented in the LDAPValueList.

Parameters **items** – List of new items.

Raises **ValueError** – if any of the items is already in the list.

LDAPValueList.**insert**(*idx*, *value*)

Insert a unique item at a given position.

Parameters

- **idx** (*int*) – the position.
- **value** – the new item.

Raises **ValueError** – if the *item* is not unique.

LDAPValueList.**remove**(*value*)

Remove the first item from the LDAPValueList whose value is *value*.

Parameters **value** – the item to be removed.

Raises **ValueError** – if *value* is not int the list.

LDAPValueList.**pop**(*idx=-1*)

Remove the item at the given position in the LDAPValueList, and return it. If no index is specified, pop() removes and returns the last item in the list.

Parameters **idx** (*int*) – optional index.

LDAPValueList.**clear**() → None

Remove all items from the LDAPValueList.

LDAPValueList.**copy**() → src.bonsai.ldapvaluelist.LDAPValueList

Return a shallow copy of the LDAPValueList. This includes the status and the previously added and deleted items.

Return type *LDAPValueList*

Returns The copy of the LDAPValueList.

LDAPValueList.**status**

The status of the LDAPValueList. The status can be:

- 0: unchanged.
- 1: added or deleted item to list.
- 2: replaced the entire list.

2.4.2 bonsai.active_directory

ACE

class bonsai.active_directory.**ACE**(*ace_type*, *flags*, *mask*, *trustee_sid*, *object_type*, *inherited_object_type*, *application_data*)

A class for the access control entry, that encodes the user rights afforded to a principal.

Parameters

- **ace_type** (ACEType) – the type of the ACE.
- **flags** (*Set[*ACEFlag*]*) – the set of flags for the ACE.
- **mask** (*int*) – the access mask to encode the user rights as an int.
- **trustee_sid** (SID) – the SID of the trustee.
- **object_type** (*uuid.UUID|None*) – a UUID that identifies a property set, property, extended right, or type of child object.
- **inherited_object_type** (*uuid.UUID|None*) – a UUID that identifies the type of child object that can inherit the ACE.
- **application_data** (*bytes*) – optional application data.

Note: For more details, see offical Microsoft docs about ACE.

classmethod ACE.**from_binary**(*data*)

Create an ACE object from a binary blob.

Parameters **data** (*bytes*) – a little-endian byte ordered byte input.

Returns A new ACE instance.

Return type *ACE*

Raises

- **TypeError** – when the parameter is not bytes.

- **ValueError** – when the input cannot be parsed as an ACE object.

ACE.**to_binary**()

Convert ACE object to binary form with little-endian byte order.

Returns Bytes of the binary ACE instance

Return type bytes

ACE.**__str__**()

Return the SDDL string representation of the ACE object.

ACE.**application_data**

The possible application data.

ACE.**flags**

The flags of the ACE.

ACE.**inherited_object_type**

The uuid of the inherited object type.

ACE.**mask**

The acces mask

ACE.**object_type**

The uuid of the object type.

ACE.**size**

The binary size of ACE in bytes.

ACE.**trustee_sid**

The sid of the trustee.

ACEFlag

class bonsai.active_directory.**ACEFlag**

ACE type-specific control flags.

OBJECT_INHERIT = 1

CONTAINER_INHERIT = 2

NO_PROPAGATE_INHERIT = 4

INHERIT_ONLY = 8

INHERITED = 16

SUCCESSFUL_ACCESS = 64

FAILED_ACCESS = 128

short_name

The SDDL short name of the flag.

ACERight

class bonsai.active_directory.**ACERight**

The rights of the ACE.

GENERIC_READ = 2147483648

GENERIC_WRITE = 67108864

`GENERIC_EXECUTE = 536870912`

`GENERIC_ALL = 268435456`

`MAXIMUM_ALLOWED = 33554432`

`ACCESS_SYSTEM_SECURITY = 16777216`

`SYNCHRONIZE = 1048576`

`WRITE_OWNER = 524288`

`WRITE_DACL = 262144`

`READ_CONTROL = 131072`

`DELETE = 65536`

`DS_CONTROL_ACCESS = 256`

`DS_CREATE_CHILD = 1`

`DS_DELETE_CHILD = 2`

`ACTRL_DS_LIST = 4`

`DS_SELF = 8`

`DS_READ_PROP = 16`

`DS_WRITE_PROP = 32`

`DS_DELETE_TREE = 64`

`DS_LIST_OBJECT = 128`

short_name
The SDDL short name of the access right.

ACEType

class bonsai.active_directory.**ACEType**
Type of the ACE.

`ACCESS_ALLOWED = 0`

`ACCESS_DENIED = 1`

`SYSTEM_AUDIT = 2`

`SYSTEM_ALARM = 3`

`ACCESS_ALLOWED_COMPOUND = 4`

`ACCESS_ALLOWED_OBJECT = 5`

`ACCESS_DENIED_OBJECT = 6`

`SYSTEM_AUDIT_OBJECT = 7`

`SYSTEM_ALARM_OBJECT = 8`

`ACCESS_ALLOWED_CALLBACK = 9`

`ACCESS_DENIED_CALLBACK = 10`

`ACCESS_ALLOWED_CALLBACK_OBJECT = 11`

`ACCESS_DENIED_CALLBACK_OBJECT = 12`

SYSTEM_AUDIT_CALLBACK = 13

SYSTEM_ALARM_CALLBACK = 14

SYSTEM_AUDIT_CALLBACK_OBJECT = 15

SYSTEM_ALARM_CALLBACK_OBJECT = 16

SYSTEM_MANDATORY_LABEL = 17

SYSTEM_RESOURCE_ATTRIBUTE = 18

SYSTEM_SCOPED_POLICY_ID = 19

short_name
The SDDL short name of the type.

is_object_type
Flag for ACE types with objects.

ACL

class `bonsai.active_directory.`**ACL**(*revision*, *aces*)
The access control list (ACL) is used to specify a list of individual access control entries (ACEs). An ACL and an array of ACEs comprise a complete access control list.

Parameters

- **revision** (`ACLRevision`) – the revision of the ACL.
- **aces** (*List*[`ACE`]) – list of *ACE*.

Note: For more details, see offical Microsoft docs about ACL.

classmethod `ACL.`**from_binary**(*data*)
Create an ACL object from a binary blob.

Parameters **data** (*bytes*) – a little-endian byte ordered byte input.

Returns A new ACL instance.

Return type *ACL*

Raises

- **TypeError** – when the parameter is not bytes.
- **ValueError** – when the input cannot be parsed as an ACL object.

`ACL.`**to_binary**()
Convert ACL object to binary form with little-endian byte order.

Returns Bytes of the binary ACL instance

Return type bytes

`ACL.`**aces**
The list of *ACE* objects.

`ACL.`**revision**
The revision of ACL.

`ACL.`**size**
The binary size in bytes.

ACLRevision

class bonsai.active_directory.**ACLRevision**

The ACL revision.

ACL_REVISION = 2

ACL_REVISION_DS = 4

SecurityDescriptor

class bonsai.active_directory.**SecurityDescriptor**(*control*, *owner_sid*, *group_sid*, *sacl*, *dacl*, *revision = 1*, *sbz1 = 0*)

A class that defines the security attributes of an object. These attributes specify who owns the object; who can access the object and what they can do with it; what level of audit logging can be applied to the object; and what kind of restrictions apply to the use of the security descriptor. It can contain two types of *ACL*:

- A discretionary access control list (DACL) is controlled by the owner of an object or anyone granted WRITE_DAC access to the object. It specifies the access particular users and groups can have to an object. For example, the owner of a file can use a DACL to control which users and groups can and cannot have access to the file.
- A system access control list (SACL) is similar to the DACL, except that the SACL is used to audit rather than control access to an object. When an audited action occurs, the operating system records the event in the security log. Each ACE in a SACL has a header that indicates whether auditing is triggered by success, failure, or both; a SID that specifies a particular user or security group to monitor; and an access mask that lists the operations to audit.

Parameters

- **control** (*dict*) – a dict that specifies control access bit flags.
- **owner_sid** (*SID | None*) – the SID of the owner of the object.
- **group_sid** (*SID | None*) – the SID of the group of the object.
- **sacl** (*ACL | None*) – the system access control list.
- **dacl** (*ACL | None*) – the discretionary access control list.
- **revision** (*int*) – the revision of the security descriptor.
- **sbz1** (*int*) – reserved value.

Note: For more details, see offical Microsoft docs about SECURITY_DESCRIPTOR.

classmethod SecurityDescriptor.**from_binary**(*data*)

Create a SecurityDescriptor object from a binary blob.

Parameters **data** (*bytes*) – a little-endian byte ordered byte input.

Returns A new SecurityDescriptor instance.

Return type *SecurityDescriptor*

Raises

- **TypeError** – when the parameter is not bytes.
- **ValueError** – when the input cannot be parsed as a SecurityDescriptor object.

SecurityDescriptor.**to_binary**()

Convert SecurityDescriptor object to binary form with little-endian byte order.

Returns Bytes of the binary SecurityDescriptor instance

Return type bytes

SecurityDescriptor.**control**

The dict of the control access bit flags.

SecurityDescriptor.**dacl**

The discretionary *ACL*.

SecurityDescriptor.**group_sid**

The *SID* of the group.

SecurityDescriptor.**owner_sid**

The *SID* of the owner.

SecurityDescriptor.**revision**

The revision of the security descriptor.

SecurityDescriptor.**sacl**

The system *ACL*.

SecurityDescriptor.**sbz1**

Reserved field in the security descriptor structure.

SID

class bonsai.active_directory.**SID**(*str_rep = None*, *bytes_le = None*)

A class for representing a Security Identifier, that identifies users, groups, and computer accounts on a Microsoft Windows platform.

Parameters

- **str_rep** (*str | None*) – a string representation of a SID.
- **bytes_le** (*bytes | None*) – a bytes representation of a SID in little-endian byte order.

Raises

- **TypeError** – when the type of the parameters are invalid, or both parameters are given.
- **ValueError** – when the given parameter cannot be parsed as a valid SID.

Note: For more details, see offical Microsoft docs about SID.

SID.**__eq__**(*other*)

Check equality of two SIDs by their identifier_authority and list of subauthorities, or if the other object is a string than by their string formats.

SID.**__repr__**()

The representation of SID class.

SID.**__str__**()

Return the string format of the SID.

SID.**bytes_le**

The byte format of the SID in little-endian byte order.

SID.**identifier_authority**
The indentifier that indicates the authority under which the SID was created.

SID.**revision**
The revision level of the SID.

SID.**sddl_alias**
The string SDDL alias of the SID if it exists, otherwise it's None.

SID.**size**
The binary size of the SID in bytes.

SID.**subauthorities**
A tuple of subauthorities that uniquely identifies a principal relative to the identifier authority.

UserAccountControl

class bonsai.active_directory.**UserAccountControl**(*flags*)
A class for parsing UserAccountControl field.

Parameters **flags** (*int*) – integer representing the property flags.

Raises **TypeError** – if flags parameter is not an int.

Note: For more details, see offical Microsoft docs about UserAccountControl.

UserAccountControl.**properties**
Dictionary of the UserAccountControl properties.

UserAccountControl.**value**
The intger value of the properties.

2.4.3 bonsai.asyncio

AIOLDAPConnection

class bonsai.asyncio.**AIOLDAPConnection**(*client*, *loop=None*)
Asynchronous LDAP connection object that works with asyncio. It has the same methods and properties as *bonsai.LDAPConnection*, but with the exception of *bonsai.LDAPConnection.close()* and *bonsai.LDAPConnection.fileno()* all of them are awaitable.

Parameters

- **client** (LDAPClient) – a client object.
- **loop** – an asyncio IO loop.

Note: The default asyncio event loop is changed with Python 3.8 on Windows to *ProactorEventLoop*. Unfortunately, bonsai's asynio connection requires the old *SelectorEventLoop*. Make sure to change it back before using the module:

```
if sys.platform == 'win32':
    import asyncio
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())
```

Getting NotImplementedError from the add_reader method of the event loop may indicate that it has not been properly set.

AIOConnectionPool

class bonsai.asyncio.**AIOConnectionPool**(*client: LDAPClient*, *minconn: int = 1*, *maxconn: int = 10*, *loop=None*, ***kwargs*)

A connection pool that can be used with asnycio tasks. It's inherited from *bonsai.pool.ConnectionPool*.

Parameters

- **client** (LDAPClient) – the *bonsai.LDAPClient* that's used to create connections.
- **minconn** (*int*) – the minimum number of connections that's created after the pool is opened.
- **maxconn** (*int*) – the maximum number of connections in the pool.
- ****kwargs** – additional keyword arguments that are passed to the *bonsai.LDAPClient.connect()* method.

Raises ValueError – when the minconn is negative or the maxconn is less than the minconn.

2.4.4 bonsai.gevent

GeventLDAPConnection

class bonsai.gevent.**GeventLDAPConnection**(*client: LDAPClient*)

Asynchronous LDAP connection object that works with Gevent. It has the same methods and properties as *bonsai.LDAPConnection*.

Parameters client (LDAPClient) – a client object.

2.4.5 bonsai.ldif

LDIFReader

class bonsai.**LDIFReader**(*input_file*, *autoload=True*, *max_length=76*)

Create an object for reading LDAP entries from an LDIF format file as described in RFC 2849.

Parameters

- **input_file** (*TextIO*) – a file-like input object in text mode.
- **autoload** (*bool*) – allow to automatically load external sources from URL.
- **max_length** (*int*) – the maximal line length of the LDIF file.

Raises TypeError – if the input_file is not a file-like object or max_length is not an int.

Example of reading an LDIF file:

```python
import bonsai

with open("./users.ldif") as fileobj:
    reader = bonsai.LDIFReader(fileobj)
    for entry in reader:
        print(entry.dn)
```

LDIFReader.**autoload**

Enable/disable autoloading resources in LDIF files.

LDIFReader.**input_file**

The file-like object of an LDIF file.

LDIFReader.**resource_handlers**

A dictionary of supported resource types. The keys are the schemes, while the values are functions that expect the full URL parameters and return the loaded content in preferably bytes format.

LDIFWriter

class bonsai.**LDIFWriter**(*output_file*, *max_length=76*)

Create an object for serialising LDAP entries in LDIF format as described in RFC 2849.

Parameters

- **output_file** (*TextIO*) – a file-like output object in text mode.
- **max_length** (*int*) – the maximal line length of the LDIF file.

Raises TypeError – if the output_file is not a file-like object or max_length is not an int.

LDIFWriter.**write_entry**(*entry*)

Write an LDAP entry to the file in LDIF format.

Parameters entry (LDAPEntry) – the LDAP entry to serialise.

```
>>> import bonsai
>>> output = open("./out.ldif", "w")
>>> writer = bonsai.LDIFWriter(output)
>>> entry = bonsai.LDAPEntry("cn=test")
>>> entry["cn"] = "test"
>>> writer.write_entry(entry)
```

LDIFWriter.**write_entries**(*entries*, *write_version=True*)

Write multiple LDAP entry to file in LDIF format, separated with newline and with optional version header.

Parameters

- **entries** (*list*) – list of LDAP entries.
- **write_version** (*bool*) – if it's True, write version header.

```
>>> client = bonsai.LDAPClient()
>>> conn = client.connect()
>>> res = conn.search("ou=nerdherd,dc=bonsai,dc=test", bonsai.LDAPSearchScope.ONE)
>>> output = open("./out.ldif", "w")
>>> writer = bonsai.LDIFWriter(output)
>>> writer.write_entries(res)
```

LDIFWriter.**write_changes**(*entry*)

Write an LDAP entry's changes to file in an LDIF-CHANGE format. Only attribute modifications are serialised.

Parameters **entry** (LDAPEntry) – the LDAP entry to serialise.

LDIFWriter.**output_file**

The file-like object for an LDIF file.

2.4.6 bonsai.pool

ConnectionPool

class bonsai.pool.**ConnectionPool**(*client: LDAPClient*, *minconn: int = 1*, *maxconn: int = 10*, ***kwargs*)

A connection pool object for managing multiple open connections.

Parameters

- **client** (LDAPClient) – the *bonsai.LDAPClient* that's used to create connections.
- **minconn** (*int*) – the minimum number of connections that's created after the pool is opened.
- **maxconn** (*int*) – the maximum number of connections in the pool.
- ****kwargs** – additional keyword arguments that are passed to the *bonsai.LDAPClient.connect()* method.

Raises **ValueError** – when the minconn is negative or the maxconn is less than the minconn.

```
>>> import bonsai
>>> from bonsai.pool import ConnectionPool
>>> client = bonsai.LDAPClient()
>>> pool = ConnectionPool(client, 1, 2)
>>> pool.open()
>>> conn = pool.get()
>>> conn.whomai()
'anonymus'
>>> pool.put(conn)
>>> pool.close()
```

ConnectionPool.**close**() → None

Close the pool and all of its managed connections.

ConnectionPool.**get**()

Get a connection from the connection pool.

Raises

- ***EmptyPool*** – when the pool is empty.
- ***ClosedPool*** – when the method is called on a closed pool.

Returns an LDAP connection object.

ConnectionPool.**open**() → None

Open the connection pool by initialising the minimal number of connections.

ConnectionPool.**put**(*conn*) → None

Put back a connection to the connection pool.

Parameters **conn** (LDAPConnection) – the connection managed by the pool.

Raises

- ***ClosedPool*** – when the method is called on a closed pool.

- ***PoolError*** – when tying to put back an object that's not managed by this pool.

`ConnectionPool.`**`spawn`**(**args*, ***kwargs*)

Context manager method that acquires a connection from the pool and returns it on exit. It also opens the pool if it hasn't been opened before.

Params *args the positional arguments passed to *bonsai.pool.ConnectionPool.get()*.

Params **kwargs the keyword arguments passed to *bonsai.pool.ConnectionPool.get()*.

Example usage:

```
import bonsai
from bonsai.pool import ConnectionPool

client = bonsai.LDAPClient()
pool = ConnectionPool(client)
with pool.spawn() as conn:
    print(conn.whoami())
```

ThreadedConnectionPool

class `bonsai.pool.`**`ThreadedConnectionPool`**(*client: LDAPClient*, *minconn: int = 1*, *maxconn: int = 10*, *block: bool = True*, ***kwargs*)

A connection pool that can be shared between threads. It's inherited from *bonsai.pool.ConnectionPool*.

Parameters

- **client** (LDAPClient) – the *bonsai.LDAPClient* that's used to create connections.
- **minconn** (*int*) – the minimum number of connections that's created after the pool is opened.
- **maxconn** (*int*) – the maximum number of connections in the pool.
- **block** (*bool*) – when it's True, the get method will block when no connection is available in the pool.
- ****kwargs** – additional keyword arguments that are passed to the *bonsai.LDAPClient.connect()* method.

Raises ValueError – when the minconn is negative or the maxconn is less than the minconn.

`ThreadedConnectionPool.`**`get`**(*timeout: Optional[float] = None*)

Get a connection from the connection pool.

Parameters timeout (*float*) – a timeout until waiting for free connection.

Raises

- ***EmptyPool*** – when the pool is empty.
- ***ClosedPool*** – when the method is called on a closed pool.

Returns an LDAP connection object.

2.4.7 bonsai.tornado

TornadoLDAPConnection

2.4.8 _bonsai

ldapsearchiter

Helper class for paged search result.

ldapsearchiter.**acquire_next_page**()

Request the next page of result. Returns with the message ID of the search operation. This method can only be used if the *LDAPClient.auto_page_acquire* is *False*.

Returns an ID of the next search operation.

Return type int.

2.4.9 Errors

class bonsai.**LDAPError**

General LDAP error.

class bonsai.**LDIFError**

General exception that is raised during reading or writing an LDIF file.

class bonsai.**AffectsMultipleDSA**

Raised, when multiple directory server agents are affected.

class bonsai.**AlreadyExists**

Raised, when try to add an entry and it already exists in the dictionary.

class bonsai.**AuthenticationError**

Raised, when authentication is failed with the server.

class bonsai.**AuthMethodNotSupported**

Raised, when the chosen authentication method is not supported.

class bonsai.**ConnectionError**

Raised, when client is not able to connect to the server.

class bonsai.**ClosedConnection**

Raised, when try to perform LDAP operation with closed connection.

class bonsai.**InsufficientAccess**

Raised, when the user has insufficient access rights.

class bonsai.**InvalidDN**

Raised, when dn string is not a valid distinguished name.

class bonsai.**InvalidMessageID**

Raised, when try to get the result with a message ID that belongs to an unpending or already finished operation.

class bonsai.**NoSuchAttribute**

Raised, when the given attribute of an entry does not exist.

class bonsai.**NoSuchObjectError**

Raised, when operation (except search) is performed on an entry that is not found in the directory.

class bonsai.**NotAllowedOnNonleaf**

Raised, when the operation is not allowed on a nonleaf object.

class bonsai.**ObjectClassViolation**
Raised, when try to add or modify an LDAP entry and it violates the object class rules.

class bonsai.**ProtocolError**
Raised, when protocol error is happened.

class bonsai.**SizeLimitError**
Raised, when the search operation exceeds the client side size limit or server side size limit that's applied to the bound user.

class bonsai.**TimeoutError**
Raised, when the specified timeout is exceeded.

class bonsai.**TypeOrValueExists**
Raised, when the attribute already exists or the value has been already assigned.

class bonsai.**UnwillingToPerform**
Raised, when the server is not willing to handle requests.

class bonsai.**PasswordPolicyError**
General exception for password policy errors.

class bonsai.**AccountLocked**
Raised, when the password policy is set, available on the server and the user's account is locked.

class bonsai.**ChangeAfterReset**
Raised, when the password policy is set, available on the server and it signifies that the password must be changed before the user will be allowed to perform any operation (except bind and modify).

class bonsai.**InsufficientPasswordQuality**
Raised, when the password policy is set, available on the server and the user's password is not strong enough.

class bonsai.**MustSupplyOldPassword**
Raised, when the password policy is set, available on the server and the existing password is not specified.

class bonsai.**PasswordExpired**
Raised, when the password policy is set, available on the server and the user's password is expired.

class bonsai.**PasswordInHistory**
Raised, when the password policy is set, available on the server and the user's password is in the history.

class bonsai.**PasswordModNotAllowed**
Raised, when the password policy is set, available on the server and the user is restricted from changing her password.

class bonsai.**PasswordTooShort**
Raised, when the password policy is set, available on the server and the user's password is too short to be set.

class bonsai.**PasswordTooYoung**
Raised, when the password policy is set, available on the server and the user's password is too young to be modified.

class bonsai.pool.**PoolError**
Connection pool related errors.

class bonsai.pool.**ClosedPool**
Raised, when the connection pool is closed.

class bonsai.pool.**EmptyPool**
Raised, when the connection pool is empty.

2.4.10 Utility functions

bonsai.utils.**escape_attribute_value**(*attrval*)

Escapes the special character in an attribute value based on RFC 4514.

Parameters **attrval** (*str*) – the attribute value.

Returns The escaped attribute value.

Return type str

```
>>> import bonsai
>>> bonsai.escape_attribute_value(",cn=escaped")
'\\,cn\\=escaped'
```

bonsai.utils.**escape_filter_exp**(*filter_exp*)

Escapes the special characters in an LDAP filter based on RFC 4515.

Parameters **filter_exp** (*str*) – the unescaped filter expression.

Returns the escaped filter expression.

Return type str

```
>>> import bonsai
>>> bonsai.escape_filter_exp("(objectclass=*)")
'\\28objectclass=\\2A\\29'
```

bonsai.**get_tls_impl_name**()

Return the identification of the underlying TLS implementation that is used by the LDAP library:

```
>>> bonsai.get_tls_impl_name()
"MozNSS"
```

The possible return values are: *GnuTLS*, *OpenSSL*, *MozNSS* and *SChannel*.

Returns A identification of TLS implementation.

Return type str

bonsai.**get_vendor_info**()

Return the vendor's name and the version number of the LDAP library:

```
>>> bonsai.get_vendor_info()
("OpenLDAP", 20440)
```

Returns A tuple of the vendor's name and the library's version.

Return type tuple

bonsai.**has_krb5_support**()

Returns True if the module is built with the optional Kerberos/GSSAPI headers.

Return type bool

bonsai.**set_connect_async**(*allow*)

Disable/enable asynchronous connection for the underlying socket, which means that the socket is set to be non-blocking when it's enabled. The default setting is *False* on every platform. This is an OpenLDAP specific setting (see *LDAP_OPT_CONNECT_ASYNC* option in the OpenLDAP documentation for further details).

Parameters **allow** (*bool*) – Enabling/disabling async connect mode.

> **Warning:** Experience shows that this is a delicate setting. Even with a newer OpenLDAP, the TLS library version used by libldap might be unable to handle non-blocking sockets correctly.

`bonsai.`**`set_debug`**(*debug*, *level=0*)

Set debug mode for the module. Turning it on will provide traceback information of C function calls on the standard output.

If the module uses OpenLDAP, then setting the *level* parameter to a non-zero integer will also give additional info about the libldap function calls.

Parameters

- **`debug`** (*`bool`*) – Enabling/disabling debug mode.
- **`level`** (*`int`*) – The debug level (for OpenLDAP only).

2.5 Changelog

2.5.1 [1.3.0 - 2021-08-24]

Changed

- Drop support for Python 3.5, the module requires at least 3.6.
- Allow space in LDAPDN's attribute type. (Issue #52)
- LDAPClient.server_chase_referrals is set to False by default.
- Preventing to set both password and keytab as credentials.

Added

- New ignore_referrals attribute for LDAPClient to be able to simply skip LDAP referrals in search results.
- New submodule for working with Active Directory: ACE, ACL, SecurityDescriptor and UserAccountControl classes.

Fixed

- Deadlock when waiting for finishing the init thread on macOS.

2.5.2 [1.2.1 - 2020-12-31]

Changed

- Asynchronous connection process during socket initialisation is disabled by default on every platform. It caused TLS errors on newer Ubuntu (18.04+). Use set_connect_async explicitly to enable it.
- LDAPClient's server_chase_referrals is set to False automatically before starting a paged_search.
- Using urllib.parse for LDAPURL, instead of regular expressions.

Fixed

- Freeing resources allocated by the LDAP structure during LDAPConnection's deallocation. (Issue #45)
- Package name in documentation. (Thanks to @ikreb7)

2.5.3 [1.2.0] - 2020-01-18

Added

- The get_result coroutines to AIOLDAPConnection and TornadoLDAPConnection.
- LDAPI example to docs. (Thanks to @senfomat)

Fixed

- Raising ConnectionError instead of ValueError of invalid file descriptor with AIOLDAPConnection when the server is unreachable. (Issue #27)
- Raising SizeLimitError when the query hits either the client-side or the server-side limit, fix condition check when acquiring next page. (Issue #31)
- Race condition for open method of ThreadedConnectionPool and AIOConnectionPool.

2.5.4 [1.1.0] - 2019-04-06

Changed

- Drop support for Python 3.4. From further releases 3.4 related codes will be removed (some asyncio related code has already changed), and the module will require 3.5 or newer Python to be built.
- Add gevent and tornado as extra requirements for setup.py.

Added

- New set_connect_async function to disable/enable asynchronous connection process during socket initialisation. (Thanks to @tck42)
- New connection pool classes: simple ConnectionPool, ThreadedConnectionPool that can be shared between threads and AIOConnectionPool for asyncio tasks.

Fixed

- Defining PY_SSIZE_T_CLEAN and changing parameter size variables from int to Py_ssize_t for Python 3.8.

2.5.5 [1.0.0] - 2018-09-09

Changed

- Separate basic search functionality to three different methods in LDAPConnection: search, paged_search and virtual_list_search.

- LDAPEntry's DN is listed among its attributes under the dn key.
- LDAPClient's set_credentials method uses optional named parameters instead of tuples.
- LDAPClient's credentials property returns a dict instead of a tuple.
- LDAPURL's filter property and the filter parameters of LDAPConnection's search methods are renamed to filter_exp.
- The representation of LDAPEntry honours the last call of its change_attribute method better than previously.
- Drop Heimdal support for advanced Kerberos capabilities (at least temporarily).
- The get_tls_impl_name, get_vendor_info, has_krb5_support, and set_debug functions are moved to the utils submodule.

Added

- LDIFReader and LDIFWriter objects for handling LDIF format.
- The delete_old_rdn parameter for LDAPEntry's rename method. (Issue #17)
- Kerberos keytab support for set_credentials (Thanks to @Mirraz).
- Utils submodule with escape_filter_exp and escape_attribute_value functions. (Issue #18)
- An exclude_dn keyword-only argument to LDAPEntry's keys, items and values methods to exclude the dn key and value from the return values.
- Support for ldapi connection in LDAPURL and LDAPConnection.
- BaseLDAPConnection as a super class for all connection classes.
- Type annotations for several methods.

Fixed

- Several reference counting errors that caused memory leaks. (Issue #19)
- Escaping brackets in LDAPURL's regular expressions. (Issue #22)
- Missing ManageDsaIT control during LDAPConnection's delete.
- Honouring timeout settings for network connections, but only on Linux with newer OpenLDAP than 2.4.43. (Issue #21)
- Typo in documentation (Thanks to @magnuswatn).

2.5.6 [0.9.1] - 2017-12-03

Changed

- LDAPError messages have the original LDAP error codes.
- TLS initialisation is separated from LDAP struct initialisation.

Added

- Async with support for AIOLDAPConnection. (Issue #12)
- New set_debug module function that enables debug mode.

Fixed

- Signalling after LDAP initialisation is failed.
- Using TLS settings while getting the root DSE in get_rootDSE method.

2.5.7 [0.9.0] - 2017-02-15

Changed

- Python 3.3 is no longer considered to be supported. The package won't be tested with 3.3 anymore.
- The LDAPSearchIter object is automatically acquiring the next page during iteration for paged LDAP search by default.
- Installing the package from source on Mac OS X became simpler with setup.cfg (Thanks to @LukeXuan).
- When recursive is True, LDAPConnection.delete uses LDAP_SERVER_TREE_DELETE control (if it is possible).
- LDAPClient.url property became writeable.

Added

- LDAPClient.set_auto_page_acquire and auto_page_acquire property for enabling/disabling automatic page acquiring during paged LDAP search.
- LDAPEntry.change_attribute and LDAPEntry.clear_attribute_changes methods for handling LDAP attributes with explicit modification operation types.
- Async iterator (async for) support for LDAPSearchIter.
- LDAPClient.server_chase_referrals property to set chasing LDAP referrals by the server.
- LDAPReference object for handling LDAP referrals.
- LDAPURL.__eq__ method to check LDAPURL objects and string equality.
- LDAPClient.set_url method to set url attribute.
- LDAPClient.set_managedsait method to support LDAP ManageDsaIT control during search, add and modify operations.

Fixed

- The value validation of LDAPDN's __setitem__ method.
- The missing asyncio.coroutine decorators of AIOLDAPConnection's methods.
- IPv6 parsing for LDAPURL.

2.5.8 [0.8.9] - 2016-11-19

Changed

- Reimplemented LDAPValueList in Python, removed C implementations of ldapvaluelist and uniquelist.
- Reimplemented LDAPEntry.delete method in Python.
- LDAPConnection.search method to accept bytes-like object as a filter parameter. (Issue #7)
- LDAPClient.get_rootDSE method uses anonym bind without any previously set LDAP controls to search for rootDSE.

Added

- LDAP_EXTENDED_DN_CONTROL support with LDAPClient.set_extended_dn method and LDAPEntry's new extended_dn string attribute. (Issue #6)

Fixed

- Case sensitivity when checking LDAPDN equality.

2.5.9 [0.8.8] - 2016-07-19

Changed

- LDAPDN object is loaded for the C extension after initialisation once, rather than loading it for every time when an LDAPEntry's DN is set.

Added

- Password policy control support with LDAPClient.set_password_policy on Unix.
- New exceptions for password policy errors.
- LDAP Password Modify extended operation support with LDAPConnection.modify_password.

Fixed

- AIOLDAPConnection hanging on write events during selecting socket descriptors.

2.5.10 [0.8.7] - 2016-06-27

Changed

- LDAPDN object to validate with regex instead of splitting to tuples.

Added

- Optional *recursive* bool parameter for LDAPConnection.delete method to remove entities in a subtree recursively.

Fixed

- Wrong typing for LDAPConnection.search when VLV is set.
- Py_None return values in C functions.
- Timeout parameter for operations of Tornado and Asyncio connections.

2.5.11 [0.8.6] - 2016-06-05

Changed

- AttributeErrors to Type- and ValueErrors for invalid function parameters.
- LDAPConnection.delete and LDAPEntry.rename accept LDAPDN as DN parameter.

Added

- New SizeLimitError.
- Some typing info and typing module dependency for 3.4 and earlier versions.

Fixed

- Ordered search returning with list (instead of ldapsearchiter).
- Setting error messages on Unix systems.
- Timeout for connecting.
- Setting default ioloop for TornadoLDAPConnection (Thanks to @lilydjwg).

2.5.12 [0.8.5] - 2016-02-23

Changed

- Removed LDAPConnection's set_page_size and set_sort_order method.
- If virtual list view parameters are set for the search, the search method will return a tuple of the results and a dictionary of the received VLV response LDAP control.
- Renamed LDAPConnection's async attribute and LDAPClient.connect method's async parameter to is_async.
- Improved Mac OS X support: provide wheel with newer libldap libs.

Added

- New optional parameters for LDAPConnection's search method to perform searches with virtual list view, paged search result and sort order.
- New module functions: get_vendor_info and get_tls_impl_name.
- NTLM and GSS-SPNEGO support for MS Windows.
- Automatic TGT requesting for GSSAPI/GSS-SPNEGO, if the necessary credential information is provided. (Available only if optional Kerberos headers are installed before building the module.)

- LDAPSearchScope enumeration for search scopes.

Fixed

- Parsing result of an extended operation, if it is not supported by the server.
- Binary data handling.
- LDAPEntry's rename method do not change the entry's DN after failure.

2.5.13 [0.8.1] - 2015-10-27

Changed

- Renamed LDAPConnection's cancel method to abandon.

Added

- Timeout support for opening an LDAP connection.

Fixed

- Possible deadlock (by constantly locking from the main thread) during initialising an LDAP session on Linux.

2.5.14 [0.8.0] - 2015-10-17

Changed

- New module name (from PyLDAP) to avoid confusion with other Python LDAP packages.
- LDAPEntry's clear and get method are rewritten in Python.
- Connection settings are accessible via properties of LDAPClient.
- Moved asyncio related code into a separate class that inherits from LDAPConnection.
- Default async class can be change to other class implementation that can work with non-asyncio based approaches (e.g. like Gevent).
- Names of the objects implemented in C are all lower-cased.

Added

- Full unicode (UTF-8) support on MS Windows with WinLDAP.
- LDAPConnection.fileno() method to get the socket descriptor of the connection.
- New methods for LDAPClient to set CA cert, client cert and client key.
- EXTERNAL SASL mechanism for binding.
- Use of authorization ID during SASL binding.
- New classes for supporting Gevent and Tornado asynchronous modules.
- Timeout parameter for LDAP operations.

Fixed

- Own error codes start from -100 to avoid overlap with OpenLDAP's and WinLDAP's error codes.
- New folder structure prevents the interpreter to try to load the local files without the built C extension(, if the interpreter is started from the module's root directory).

2.5.15 [0.7.5] - 2015-07-12

Changed

- LDAPClient.connect is a coroutine if async param is True. (Issue #1)
- The binding function on Windows uses ldap_sasl_bind instead of the deprecated ldap_bind.
- The connection procedure (init, set TLS, bind) creates POSIX and Windows threads to avoid I/O blocking.
- Optional error messages are appended to the Python LDAP errors.

Added

- New open method for LDAPConnection object to build up the connection.
- New LDAPConnectIter object for initialisation, setting TLS, and binding to the server.

Fixed

- LDAPConnection.whoami() returns 'anonymous' after an anonymous bind.
- After failed connection LDAPClient.connect() returns ConnectionError on MS Windows.

2.5.16 [0.7.0] - 2015-01-28

Changed

- The set_page_size method is moved from LDAPClient to LDAPConnection.

Added

- Support for asynchronous LDAP operations.
- Cancel method for LDAPConnection.
- New LDAPEntry and LDAPConnection Python objects as wrappers around the C implementations.

Fixed

- UniqueList contains method.

2.5.17 [0.6.0] - 2014-09-24

Changed

- LDAPClient accepts LDAPURL objects as url.
- LDAPConnection search accepts LDAPDN objects as basedn parameter.

Added

- Method to set certificate policy.
- Server side sort control.

Fixed

- Getting paged result cookie on MS Windows.
- Segmentation fault of LDAPEntry.popitem().

2.5.18 [0.5.0] - 2014-03-08

Changed

- Module name to lower case.
- Removed get_entry method.
- LDAP URL parameters are used for search properly.

Added

- New LDAPClient object for managing the connection settings.
- DIGEST-MD5 support on MS Windows.
- Raw attribute support: the given attributes will be kept in bytearray form.
- Paged search control support.
- Sphinx documentation with tutorial.

Fixed

- Several memory management issues.

2.5.19 [0.1.5] - 2013-07-31

Changed

- Errors are implemented in Python.
- Using WinLDAP on MS Windows for LDAP operations.

Added

- UniqueList for storing case-insensitive unique elements.
- LDAPURL and LDAPDN Python classes for handling LDAP URL and distinguished name.

Fixed

- Getting empty list for searching non-existing entries.

2.5.20 [0.1.0] - 2013-06-23

- Initial public release.

CHAPTER 3

Contribution

Any contributions are welcome. If you would like to help in development fork or report issue on the project's GitHub site. You can also help in improving the documentation.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index