
dataflake.Idapconnection Documentation

Release 1.5

Jens Vagelpohl

April 20, 2012

CONTENTS

1	Narrative documentation	3
1.1	Installation	3
1.2	Using dataflake.ldapconnection	3
1.3	Development	4
1.4	Change log	8
2	API documentation	13
2.1	Interfaces	13
2.2	<code>dataflake.ldapconnection.connection</code>	13
3	Support	15
4	Indices and tables	17
	Python Module Index	19
	Index	21

`dataflake.ldapconnection` provides an abstraction layer on top of *python-ldap*. It offers a connection object with simplified methods for inserting, modifying, searching and deleting records in the LDAP directory tree. Failover/redundancy can be achieved by supplying connection data for more than one LDAP server.

NARRATIVE DOCUMENTATION

Narrative documentation explaining how to use `dataflake.ldapconnection`.

1.1 Installation

You will need `Python` version 2.4 or better to run `dataflake.ldapconnection`. Development of `dataflake.ldapconnection` is done primarily under Python 2.7, so that version is recommended.

Warning: To successfully install `dataflake.ldapconnection`, you will need `setuptools` installed on your Python system in order to run the `easy_install` command.

It is advisable to install `dataflake.ldapconnection` into a *virtualenv* in order to obtain isolation from any “system” packages you’ve got installed in your Python version (and likewise, to prevent `dataflake.ldapconnection` from globally installing versions of packages that are not compatible with your system Python).

After you’ve got the requisite dependencies installed, you may install `dataflake.ldapconnection` into your Python environment using the following command:

```
$ easy_install dataflake.ldapconnection
```

If you use `zc.buildout` you can add `dataflake.ldapconnection` to the necessary `eggs` section to have it pulled in automatically.

When you `easy_install dataflake.ldapconnection`, the *python-ldap* libraries are installed if they are not present.

1.2 Using `dataflake.ldapconnection`

`dataflake.ldapconnection` provides an abstraction layer on top of *python-ldap*. It offers a connection object with simplified methods for inserting, modifying, searching and deleting records in the LDAP directory tree. Failover/redundancy can be achieved by supplying connection data for more than one LDAP server.

1.2.1 API examples

Instantiating a connection object:

```
1 >>> from dataflake.ldapconnection.connection import LDAPConnection
2 >>> conn = LDAPConnection()
3 >>> conn.addServer('localhost', '1389', 'ldap')
```

To work with the connection object you need to make sure that a LDAP server is available on the provided host and port.

Now we will search for a record that does not yet exist, then add the missing record and find it when searching again:

```
1 >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
2 {'exception': '', 'results': [], 'size': 0}
3 >>> data = { 'objectClass': ['top', 'inetOrgPerson']
4 ...       , 'cn': 'testing'
5 ...       , 'sn': 'Lastname'
6 ...       , 'givenName': 'Firstname'
7 ...       , 'mail': 'test@test.com'
8 ...       , 'userPassword': '5secret'
9 ...       }
10 >>> conn.insert('ou=users,dc=localhost', 'cn=testing', attrs=data, bind_dn='cn=Manager,dc=localhost')
11 >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
12 {'exception': '', 'results': [{'dn': 'cn=testing,ou=users,dc=localhost', 'cn': ['testing'], 'objectC
```

We can edit an existing record:

```
1 >>> changes = {'givenName': 'John', 'sn': 'Doe'}
2 >>> conn.modify('cn=testing,ou=users,dc=localhost', attrs=changes, bind_dn='cn=Manager,dc=localhost')
3 >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
4 {'exception': '', 'results': [{'dn': 'cn=testing,ou=users,dc=localhost', 'cn': ['testing'], 'objectC
```

As the last step, we will delete our testing record:

```
1 >>> conn.delete('cn=testing,ou=users,dc=localhost', bind_dn='cn=Manager,dc=localhost', bind_pwd='secret')
2 >>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
3 {'exception': '', 'results': [], 'size': 0}
```

The *Interfaces* page contains more information about the connection APIs.

1.2.2 Handling string encoding for input and output values

LDAP servers expect values sent to them in specific string encodings. Standards-compliant LDAP servers use UTF-8. They use the same encoding for values returned e.g. by a search. This server-side encoding may not be convenient for communicating with the `dataflake.ldapconnection` API itself. For this reason the server-side encoding and API encoding can be set individually on connection instances using the attributes `ldap_encoding` and `api_encoding`, respectively. The connection instance handles all string encoding transparently.

By default, instances use UTF-8 as `ldap_encoding` and ISO-8859-1 (Latin-1) as `api_encoding`. You can assign any valid Python codec name to these attributes. Assigning an empty value or `None` means that unencoded unicode strings are used.

1.3 Development

1.3.1 Getting the source code

The source code is maintained in the Dataflake Git repository. To check out the trunk:

```
$ git clone https://git.dataflake.org/git/dataflake.ldapconnection
```

You can also browse the code online at <http://git.dataflake.org/cgit/dataflake.ldapconnection>

1.3.2 Bug tracker

For bug reports, suggestions or questions please use the Launchpad bug tracker at <https://bugs.launchpad.net/dataflake.ldapconnection>.

1.3.3 Sharing Your Changes

Note: Please ensure that all tests are passing before you submit your code. If possible, your submission should include new tests for new features or bug fixes, although it is possible that you may have tested your new code by updating existing tests.

If you got a read-only checkout from the Git repository, and you have made a change you would like to share, the best route is to let Git help you make a patch file:

```
$ git diff > dataflake.ldapconnection-cool_feature.patch
```

You can then upload that patch file as an attachment to a Launchpad bug report.

1.3.4 Running the tests in a virtualenv

If you use the `virtualenv` package to create lightweight Python development environments, you can run the tests using nothing more than the `python` binary in a `virtualenv`. First, create a scratch environment:

```
$ /path/to/virtualenv --no-site-packages /tmp/virtualpy
```

Next, get this package registered as a “development egg” in the environment:

```
$ /tmp/virtualpy/bin/python setup.py develop
```

Finally, run the tests using the build-in `setuptools` testrunner:

```
$ /tmp/virtualpy/bin/python setup.py test
running test
...
test_escape_dn (dataflake.ldapconnection.tests.test_utils.UtilsTest) ... ok
```

```
-----
Ran 88 tests in 0.058s
```

```
OK
```

If you have the `nose` package installed in the `virtualenv`, you can use its testrunner too:

```
$ /tmp/virtualpy/bin/easy_install nose
...
$ /tmp/virtualpy/bin/python setup.py nosetests
running nosetests
```

```
.....
.....
-----
```

```
Ran 101 tests in 0.162s
```

```
OK
```

```
or:
```

```
$ /tmp/virtualpy/bin/nosetestests
```

```
.....
```

```
-----  
Ran 101 tests in 0.160s
```

```
OK
```

If you have the coverage package installed in the virtualenv, you can see how well the tests cover the code:

```
$ /tmp/virtualpy/bin/easy_install nose coverage
```

```
...
```

```
$ /tmp/virtualpy/bin/python setup.py nosetestests \  
    --with-coverage --cover-package=dataflake.ldapconnection  
running nosetestests
```

```
...
```

Name	Stmts	Exec	Cover	Missing
dataflake.ldapconnection	1	1	100%	
dataflake.ldapconnection.connection	246	244	99%	214-215
dataflake.ldapconnection.interfaces	10	10	100%	
dataflake.ldapconnection.utils	7	7	100%	
TOTAL	264	262	99%	

```
-----  
Ran 101 tests in 0.226s
```

```
OK
```

1.3.5 Building the documentation in a virtualenv

dataflake.ldapconnection uses the nifty Sphinx documentation system for building its docs. Using the same virtualenv you set up to run the tests, you can build the docs:

```
$ /tmp/virtualpy/bin/easy_install Sphinx
```

```
...
```

```
$ cd docs
```

```
$ PATH=/tmp/virtualpy/bin:$PATH make html
```

```
sphinx-build -b html -d _build/doctrees . _build/html
```

```
...
```

```
build succeeded.
```

Build finished. The HTML pages are in `_build/html`.

You can also test the code snippets in the documentation:

```
$ PATH=/tmp/virtualpy/bin:$PATH make doctest
```

```
sphinx-build -b doctest -d _build/doctrees . _build/doctest
```

```
...
```

```
running tests...
```

```

Doctest summary
=====
    0 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests in the sources finished, look at the \
    results in _build/doctest/output.txt.

```

1.3.6 Running the tests using `zc.buildout`

`dataflake.ldapconnection` ships with its own `buildout.cfg` file and `bootstrap.py` for setting up a development buildout:

```

$ python bootstrap.py
...
Generated script '../bin/buildout'
$ bin/buildout
...

```

Once you have a buildout, the tests can be run as follows:

```

$ bin/test --all
Running tests at all levels
Running zope.testing.testrunner.layer.UnitTests tests:
  Set up zope.testing.testrunner.layer.UnitTests in 0.000 seconds.
  Running:
.....
.....
  Ran 94 tests with 0 failures and 0 errors in 0.042 seconds.
Tearing down left over layers:
  Tear down zope.testing.testrunner.layer.UnitTests in 0.000 seconds.

```

1.3.7 Building the documentation using `zc.buildout`

The `dataflake.ldapconnection` buildout installs the Sphinx scripts required to build the documentation, including testing its code snippets:

```

$ bin/docbuilder.sh
rm -rf _build/*
sphinx-build -b doctest -d _build/doctrees . _build/doctest
Making output directory...
Running Sphinx v1.1.3
...
running tests...

Doctest summary
=====
    0 tests
    0 failures in tests
    0 failures in setup code
build succeeded.
Testing of doctests in the sources finished, look at the results in \
    ../docs/_build/doctest/output.txt.
.../bin/sphinx-build -b html -d ../docs/_build/doctrees \
    ../docs ../docs/_build/html

```

```
...
build succeeded.
```

Build finished. The HTML pages are in `.../docs/_build/html`.

To build the documentation as PDF you first need to ensure your system has a `latex2pdf` binary installed.

```
$ bin/pdfbuilder.sh
sphinx-build -b latex -d _build/doctrees . _build/latex
Making output directory...
Running Sphinx v1.1.3
...
Output written on dataflake.ldapconnection.pdf (23 pages, 128015 bytes).
Transcript written on dataflake.ldapconnection.log.
```

1.3.8 Making a release

These instructions assume that you have a development sandbox set up using `zc.buildout` as the scripts used here are generated by the buildout.

The first thing to do when making a release is to check that the ReST to be uploaded to PyPI is valid:

```
$ bin/docpy setup.py --long-description | bin/rst2 html \
  --link-stylesheet \
  --stylesheet=http://www.python.org/styles/styles.css > desc.html
```

Once you're certain everything is as it should be, the following will build the distribution, upload it to PyPI, register the metadata with PyPI and upload the Sphinx documentation to PyPI:

```
$ bin/buildout -o
$ bin/docbuilder.sh
$ bin/pdfbuilder.sh
$ bin/docpy setup.py sdist register upload upload_sphinx \
  --upload-dir=docs/_build/html
```

The `bin/buildout` step will make sure the correct package information is used.

1.4 Change log

1.4.1 1.5 (2012-04-20)

- Factored the `dataflake.ldapconnection.tests.fakeldap` module into a separate package `dataflake.fakeldap`
- Restricted the supported Python versions to 2.6 and 2.7.
- removed old `test_suite` fossils

1.4.2 1.4 (2012-04-03)

- Added a simple `tox` configuration to quickly test with different Python versions. As a result of successful testing, re-declare compatibility with Python 2.4 and 2.5.
- Moved the documentation build dependencies into a `setup extra` so building them without `buildout` becomes easier.

- Added `setup.py` aliases for creating a testing sandbox and the documentation dependencies, and use them in the buildout configuration.
- Extended `fakeldap.RaisingFakeLDAPConnection` to accept a list of exceptions to raise. On each call to the method that is set to raise the exception, the first item in the exception list is removed and raised. This allows testing code areas nested in more than one `try/except` clause.

1.4.3 1.3 (2012-03-23)

- Update `bootstrap.py` to what ships with `zc.buildout 1.5.2`
- `fakeldap`: Add email characters and some non-ASCII characters to `FLTR_RE`, to be able to use them in searches. (Stefan Holec)
- `fakeldap`: Add `unbind_s` API. (Stefan Holec)
- `fakeldap`: Deep-copy entries before returning them. (Stefan Holec)
- `fakeldap`: Only return requested attributes. (Stefan Holec)
- `fakeldap`: Optionally disable password hashing. (Stefan Holec)
- `fakeldap`: Optionally maintain the ‘memberOf’ attribute of group members. (Stefan Holec)
- `connection`: Clean up encoding and decoding of values for when the `api_encoding` is set to `None`. (Stefan Holec)
- `connection`: New ‘raw’ kwarg for the search API. If true, search results are returned in the `ldap_encoding`. (Stefan Holec)
- `connection`: Store a connection’s `bind_dn` and `bind_pwd` as is, and encode them before use. (Stefan Holec)
- `connection`: Fix a condition that caused rebinding to fail if only one of `bind_dn` and `bind_pwd` had changed. (Stefan Holec)
- `connection`: Add `disconnect` API. (Stefan Holec)
- `connection`: Allow to delete subsets from multi-valued attributes. (Stefan Holec)
- switched documentation to point to the new Git repository

1.4.4 1.2 (2010-08-09)

- Using `id()` is not random enough for a unique hash.

1.4.5 1.1 (2010-05-09)

- Updated Sphinx Makefile and configuration to be closer to the latest Sphinx version
- Greatly expand installation and testing documentation using ideas from Tres Seaver

1.4.6 1.0 (2010-04-12)

- Bug: `fakeldap.FakeLDAPConnection` wildcard searches did not work correctly and returned too many matches.
- Bug: Improve behavior matching of standard `python-ldap` and `fakeldap` by raising `ldap.NO_SUCH_OBJECT` where operations target non-existing entries.
- Bug: Improve behavior matching of standard `python-ldap` and `fakeldap` by raising `ldap.ALREADY_EXISTS` where operations duplicate existing entries.

- Bug: Added tests for all fakeldap.FakeLDAPConnection methods and added tests for some other module classes and functions.
- Refactoring: Removed the fakeldap.initialize and explode_dn functions. They were either not needed or needlessly duplicating existing python-ldap features.
- Bug: python-ldap will no longer support the LDAP connection class ldap.LdapObject.SmartLDAPObject with version 2.3.11. Replacing it with ReconnectLDAPObject.
- Bug: If a connection raised an LDAP exception inside start_tls_s handling was broken.
- Feature: You can now add server definitions for servers that support the StartTLS extended operation. Whereas the existing secure connections using the ldaps protocol are encrypted throughout, StartTLS is used through an unencrypted connection to request all further traffic to be encrypted.
- Refactoring: Switch tests to using the fakeldap LDAP connection object wherever possible, and correct a few fakeldap and LDAPConnection misbehaviors along the way.

1.4.7 1.0b1 (2010-02-01)

- Performing more rigorous input checking for DN's
- Made encoding/decoding more flexible by adding configuration flags for the encoding used by the LDAP server and the encoding for calls to and return values from the connection API. The default is backwards compatible (UTF-8 for the LDAP server encoding, and Latin-15 for the API encoding).
- Factored the connection tests module into a series of modules, it was getting large and unwieldy.
- move the actual python-ldap connection from an attribute into a module-level cache since those connections cannot be pickled.
- Removed the rdn_attr attribute, which was used to try and determine if a modify operation should trigger a modrdn. We now fish the RDN attribute from the record's DN for this purpose.
- Changed the way internal logging is done to avoid storing logger objects onto the connection instance unless it is explicitly specified. This means the instance is picklable when using the default logging.
- Removed the *bind* method. There was no good reason to expose it as part of the public API, and since bind operations are re-done as part of all operations it would only serve to confuse users. Users who want to use credentials other than the credentials configured into the connection instance should pass them along explicitly when invoking the operation.
- The search method now provides a default search subtree search scope if none is specified.
- Creating a new instance does not require passing server data like host, port and protocol anymore.
- replaced several methods with better alternatives from python-ldap, which also requires upping the dependency to python-ldap>=2.3.0, and fixing up the tests.
- pare down fakeldap to not try and provide all kinds of constants from python-ldap, but just a LDAP connection class.
- add a new method "bind" to rebind a connection, if the last bind differs from the desired bind.
- rename variable name "filter" with "fltr" to stop shadowing the Python function "filter".
- added an interfaces file as documentation and "contract". This adds a dependency on zope.interface.
- removed unused argument "login_attr" from constructor argument list
- LDAPConnection objects now accept more than a single server definition. Failover between connections is triggered by connection or operation timeouts. Added API to add and remove server definitions at runtime.

- all those methods causing LDAP operations to be performed accept optional `bind_dn` and `bind_pwd` named arguments to rebind with the provided credentials instead of those credentials stored in the `LDAPConnection` instance. This represents an API change for the *insert*, *modify* and *delete* methods.

1.4.8 0.4 (2008-12-25)

- fakeldap bug: the `modify_s` method would expect changes of type `MOD_DELETE` to come with a list of specific attribute values to delete. Now the attribute will be deleted as a whole if the expected list is `None`, this reflects actual `python-ldap` behavior better.
- now we are exercising the `fakeldap` doctests from within this package, they used to be run from `Products.LDAPUserFolder`, which was not cleaned up when the `fakeldap` module moved to `dataflake.ldapconnection`.

1.4.9 0.3 (2008-08-30)

- fakeldap: no longer override the LDAP exceptions, just get them from `python-ldap`. (http://www.dataflake.org/tracker/issue_00620)

1.4.10 0.2 (2008-08-27)

- backport a fix applied to the `LDAPUserFolder FakeLDAP` module to handle `BASE`-scoped searches on a `DN`.

1.4.11 0.1 (2008-06-11)

- Initial release.

API DOCUMENTATION

API documentation for `dataflake.ldapconnection`.

2.1 Interfaces

interface `dataflake.ldapconnection.interfaces.ILDAPConnection`
 LDAPConnection interface

ILDAPConnection instances provide a simplified way to talk to a LDAP server. They allow defining one or more server connections for automatic failover in case one LDAP server becomes unavailable.

2.2 `dataflake.ldapconnection.connection`

class `dataflake.ldapconnection.connection.LDAPConnection` (*host*='', *port*=389,
protocol='ldap',
c_factory=<class
ldap.ldapobject.ReconnectLDAPObject
at 0x10342e188>,
rdn_attr='', *bind_dn*='',
bind_pwd='',
read_only=False,
conn_timeout=-1,
op_timeout=-1, *log-*
ger=None)

LDAPConnection object

See *interfaces.py* for interface documentation.

addServer (*host, port, protocol, conn_timeout=-1, op_timeout=-1*)
 Add a server definition to the list of servers used

connect (*bind_dn=None, bind_pwd=None*)
 initialize an ldap server connection

This method returns an instance of the underlying *python-ldap* connection class. It does not need to be called explicitly, all other operations call it implicitly.

delete (*dn, bind_dn=None, bind_pwd=None*)
 Delete a record

disconnect ()

Unbind the connection and invalidate the cache

insert (*base, rdn, attrs=None, bind_dn=None, bind_pwd=None*)

Insert a new record

attrs is expected to be a mapping where the value may be a string or a sequence of strings. Multiple values may be expressed as a single string if the values are semicolon-delimited. Values can be marked as binary values, meaning they are not encoded as UTF-8, by appending ‘;binary’ to the key.

logger ()

Get the logger

modify (*dn, mod_type=None, attrs=None, bind_dn=None, bind_pwd=None*)

Modify a record

removeServer (*host, port, protocol*)

Remove a server definition from the list of servers used

search (*base, scope=2, fltr='(objectClass=*)', attrs=None, convert_filter=True, bind_dn=None, bind_pwd=None, raw=False*)

Search for entries in the database

SUPPORT

If you need commercial support for this software package, please contact zetwork GmbH at <http://www.zetwork.com>.

INDICES AND TABLES

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

PYTHON MODULE INDEX

d

`dataflake.ldapconnection.connection`, 13

INDEX

A

addServer() (dataflake.ldapconnection.connection.LDAPConnection method), 13

C

connect() (dataflake.ldapconnection.connection.LDAPConnection method), 13

D

dataflake.ldapconnection.connection (module), 13

delete() (dataflake.ldapconnection.connection.LDAPConnection method), 13

disconnect() (dataflake.ldapconnection.connection.LDAPConnection method), 13

I

ILDAPConnection (interface in dataflake.ldapconnection.interfaces), 13

insert() (dataflake.ldapconnection.connection.LDAPConnection method), 14

L

LDAPConnection (class in dataflake.ldapconnection.connection), 13

logger() (dataflake.ldapconnection.connection.LDAPConnection method), 14

M

modify() (dataflake.ldapconnection.connection.LDAPConnection method), 14

R

removeServer() (dataflake.ldapconnection.connection.LDAPConnection method), 14

S

search() (dataflake.ldapconnection.connection.LDAPConnection method), 14