dataflake.ldapconnection Documentation

Release 1.0

Jens Vagelpohl

April 12, 2010

CONTENTS

`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.

CHAPTER

ONE

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 2.6, 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 `egss` 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:

```
>>> from dataflake.ldapconnection.connection import LDAPConnection
>>> conn = LDAPConnection()
>>> 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:

```
>>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
{'exception': '', 'results': [], 'size': 0}
>>> data = { 'objectClass': ['top', 'inetOrgPerson']
...        , 'cn': 'testing'
...        , 'sn': 'Lastname'
...        , 'givenName': 'Firstname'
...        , 'mail': 'test@test.com'
...        , 'userPassword': '5ecret'
...        }
>>> conn.insert('ou=users,dc=localhost', 'cn=testing', attrs=data, bind_dn='cn=Manager,dc=localho
>>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
{'exception': '', 'results': [{'dn': 'cn=testing,ou=users,dc=localhost', 'cn': ['testing'], 'obje
```

We can edit an existing record:

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

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

```
>>> conn.delete('cn=testing,ou=users,dc=localhost', bind_dn='cn=Manager,dc=localhost', bind_pwd='s
>>> conn.search('ou=users,dc=localhost', fltr='(cn=testing)')
{'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 Subversion repository at http://svn.dataflake.org. To check out the trunk:

```
svn co http://svn.dataflake.org/svn/dataflake.ldapconnection/trunk/
```

You can also browse the code online at http://svn.dataflake.org/viewvc/dataflake.ldapconnection.

When using setuptools or zc.buildout you can use the following URL to retrieve the latest development code as Python egg:

```
http://svn.dataflake.org/svn/dataflake.ldapconnection/trunk#egg=dataflake.ldapconnection
```

1.3.2 Bug tracker

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

1.3.3 Setting up a development sandbox and testing

Once you've obtained a source checkout, you can follow these instructions to perform various development tasks. All development requires that you run the buildout from the package root directory:

```
$ python bootstrap.py
$ bin/buildout
```

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

```
$ bin/test
```

1.3.4 Building the documentation

The Sphinx documentation is built by doing the following from the directory containing setup.py:

```
$ cd docs
$ make html
```

1.3.5 Making a release

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 > build/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/docpy setup.py sdist register upload upload_sphinx --upload-dir=docs/_build/html
```

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

1.4 Change log

1.4.1 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.

1.4.2 1.0b1 (2010-02-01)

- Performing more rigorous input checking for DNs
- 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.

1.4.3 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.

1.4.4 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)

1.4.5 0.2 (2008-08-27)

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

1.4.6 0.1 (2008-06-11)

- Initial release.

CHAPTER

TWO

API DOCUMENTATION

API documentation for `dataflake.ldapconnection`.

2.1 Interfaces

interface dataflake.ldapconnection.interfaces.ILDAPConnection

ILDAPConnection 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.

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

Insert a new record

The record will be inserted at *base* with the new RDN *rdn*. *attrs* is expected to be a key:value 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 in the encoding specified as the server encoding before being sent to the LDAP server, by appending ';binary' to the key.

In order to perform the operation using credentials other than the credentials configured on the instance a DN and password may be passed in.

addServer(*host, port, protocol, conn_timeout=-1, op_timeout=-1*)

Add a server definition

protocol can be any one of `ldap` (unencrypted traffic), `ldaps` (encrypted traffic to a separate port), `ldaptls` (sets up encrypted traffic on the normal unencrypted port), or `ldapi` (trafic through a UNIX domain socket on the file system).

The *conn_timeout* argument defines the number of seconds to wait until a new connection attempt is considered failed, which means the next server is tried if it has been defined. -1 means "wait indefinitely",

The *op_timeout* argument defines the number of seconds to wait until a LDAP server operation is considered failed, which means the next server is tried if it has been defined. -1 means "wait indefinitely".

If a server definition with a host, port and protocol that matches an existing server definition is added, the new values will replace the existing definition.

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

Modify the record specified by the given DN

mod_type is one of the LDAP modification types as declared by the *python-ldap*-module, such as *ldap.MOD_ADD*, PUrl(urlscheme=protocol, hostport=hostport) provided, the modification type is guessed by comparing the current record with the *attrs* mapping passed in.

attrs is expected to be a key:value 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 before sending the to the LDAP server, by appending ';binary' to the key.

In order to perform the operation using credentials other than the credentials configured on the instance a DN and password may be passed in.

search(*base, scope=2, fltr='(objectClass=*)', attrs=None, convert_filter=True, bind_dn=None, bind_pwd=None*)
Perform a LDAP search

The search *base* is the point in the tree to search from. *scope* defines how to search and must be one of the scopes defined by the *python-ldap* module (*ldap.SCOPE_BASE*, *ldap.SCOPE_ONELEVEL* or *ldap.SCOPE_SUBTREE*). By default, *ldap.SCOPE_SUBTREE* is used. What to search for is described by the *filter* argument, which must be a valid LDAP search filter string. If only certain record attributes should be returned, they can be specified in the *attrs* sequence.

If the search raised no errors, a mapping with the following keys is returned:

- results: A sequence of mappings representing a matching record
- size: The number of matching records

The results sequence itself contains mappings that have a *dn* key containing the full distinguished name of the record, and key/values representing the records' data as returned by the LDAP server.

In order to perform the operation using credentials other than the credentials configured on the instance a DN and password may be passed in.

removeServer(*host, port, protocol*)
Remove a server definition

Please note: I you remove the server definition of a server that is currently being used, that connection will continue to be used until it fails or until the Python process is restarted.

connect(*bind_dn=None, bind_pwd=None*)
Return a working LDAP server connection

If no DN or password for binding to the LDAP server are passed in, the DN and password configured into the LDAP connection instance are used.

The connection is cached and will be re-used. Since a bind operation is forced every time the method can be used to re-bind the cached connection with new credentials.

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.

Raises RuntimeError if no server definitions are available. If all defined server connections fail the LDAP exception thrown by the last attempted connection is re-raised.

delete(*dn, bind_dn=None, bind_pwd=None*)
Delete the record specified by the given DN

In order to perform the operation using credentials other than the credentials configured on the instance a DN and password may be passed in.

2.2 dataflake.ldapconnection.connection

class **LDAPConnection**(*host='', port=389, protocol='ldap', c_factory=<class ldap.ldapobject.ReconnectLDAPObject at 0x102ead350>, rdn_attr='', bind_dn='', bind_pwd='', read_only=False, conn_timeout=-1, op_timeout=-1, logger=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

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*)
Search for entries in the database

CHAPTER

THREE

SUPPORT

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

CHAPTER

FOUR

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*
- *Glossary*

MODULE INDEX

D

INDEX