dataflake.ldapconnection Documentation

Release 1.0dev

Jens Vagelpohl

March 10, 2009

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 Installing `dataflake.ldapconnection`

1.1.1 How To Install

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

> **Warning:** To succesfully install `dataflake.ldapconnection`, you will need an environment capable of compiling Python C code. See the documentation about installing, e.g. `gcc` and `python-devel` for your system. You will also need *setuptools* installed on within 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
```

1.1.2 What Gets Installed

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

1.2 Development

1.2.1 Bug tracker

For bug reports, suggestions or questions please use the Dataflake bug tracker.

1.2.2 Source code

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

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

You can also browse the code online.

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 Glossary

Setuptools Setuptools builds on Python's `distutils` to provide easier building, distribution, and installation of packages.

Interface An attribute of a model object that determines its type. It is an instance of a `zope.interface` Interface class.

Zope The Z Object Publishing Framework. The granddaddy of Python web frameworks.

Virtualenv An isolated Python environment. Allows you to control which packages are used on a particular project by cloning your main Python. virtualenv was created by Ian Bicking.

python-ldap The *python-ldap <http://python-ldap.sourceforge.net>_* library is used to communicate with LDAP servers.

1.4 `dataflake.ldapconnection` Change History

```
Changelog for dataflake.ldapconnection
======================================

1.0 (unreleased)
----------------

- 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.
```

```
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.

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)


0.2 (2008-08-27)
----------------

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


0.1 (2008-06-11)
----------------

- Initial release.
```

CHAPTER

TWO

API DOCUMENTATION

API documentation for `dataflake.ldapconnection`.

2.1 dataflake.ldapconnection.interfaces

Interface: `dataflake.ldapconnection.interfaces.ILDAPConnection` ILDAPConnection interface

ILDAPConnection instances provide a simplified way to talk to a LDAP server. It allows defining one or more server connections that allow for automatic failover in case the current connection fails.

Method: `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 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.

Method: `addServer(host, port, protocol, conn_timeout=-1, op_timeout=-1)` Add a server definition

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.

Method: `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.

Method: `search(base, scope, filter='(objectClass=*)', attrs=None, convert_filter=True, b` Perform a LDAP search

The search *base* is the point in the tree to searc 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*). 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.

Method: `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 process is restarted.

Method: `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.

Raises RuntimeError if no server definitions are available.

Method: `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.

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*

INDEX