Ldaptor Documentation

Release 16.0

Tommi Virtanen and Bret Curtis

Sep 30, 2020

CONTENTS

CHAPTER

ONE

WHAT IS LDAPTOR

Ldaptor is a pure-Python Twisted library that implements:

- LDAP client and server logic
- separately-accessible LDAP and BER protocol message generation/parsing
- ASCII-format LDAP filter generation and parsing
- LDIF format data generation

Get it from PyPI, find out what's new in the *Changelog*!

1.1 Quick Start

1.1.1 LDAP Client Quickstart

```
import sys

from twisted.internet import defer
from twisted.internet.endpoints import clientFromString, connectProtocol
from twisted.internet.task import react
from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols.ldap.ldapsyntax import LDAPEntry


@defer.inlineCallbacks
def onConnect(client):
    # The following arguments may be also specified as unicode strings
    # but it is recommended to use byte strings for ldaptor objects
    basedn = b'dc=example,dc=org'
    binddn = b'cn=bob,ou=people,dc=example,dc=org'
    bindpw = b'secret'
    query = b'(cn=bob)'
    try:
        yield client.bind(binddn, bindpw)
    except Exception as ex:
        print(ex)
        raise
    o = LDAPEntry(client, basedn)
    results = yield o.search(filterText=query)
    for entry in results:
        print(entry.getLDIF())
```

(continues on next page)

(continued from previous page)

```
def onError(err):
    err.printDetailedTraceback(file=sys.stderr)


def main(reactor):
    endpoint_str = "tcp:host=127.0.0.1:port=8080"
    e = clientFromString(reactor, endpoint_str)
    d = connectProtocol(e, LDAPClient())
    d.addCallback(onConnect)
    d.addErrback(onError)
    return d


react(main)
```

1.1.2 LDAP Server Quick Start

```
import sys

try:
    from cStringIO import StringIO as BytesIO
except ImportError:
    from io import BytesIO

from twisted.application import service
from twisted.internet.endpoints import serverFromString
from twisted.internet.protocol import ServerFactory
from twisted.python.components import registerAdapter
from twisted.python import log
from ldaptor.inmemory import fromLDIFFile
from ldaptor.interfaces import IConnectedLDAPEntry
from ldaptor.protocols.ldap.ldapserver import LDAPServer

LDIF = b"""\
dn: dc=org
dc: org
objectClass: dcObject

dn: dc=example,dc=org
dc: example
objectClass: dcObject
objectClass: organization

dn: ou=people,dc=example,dc=org
objectClass: organizationalUnit
ou: people

dn: cn=bob,ou=people,dc=example,dc=org
cn: bob
gn: Bob
mail: bob@example.org
objectclass: top
objectclass: person
objectClass: inetOrgPerson
```

(continues on next page)

(continued from previous page)

```
sn: Roberts
userPassword: secret

dn: gn=John+sn=Doe,ou=people,dc=example,dc=org
objectClass: addressbookPerson
gn: John
sn: Doe
street: Back alley
postOfficeBox: 123
postalCode: 54321
postalAddress: Backstreet
st: NY
l: New York City
c: US
userPassword: terces

dn: gn=John+sn=Smith,ou=people, dc=example,dc=org
objectClass: addressbookPerson
gn: John
sn: Smith
telephoneNumber: 555-1234
facsimileTelephoneNumber: 555-1235
description: This is a description that can span multi
 ple lines as long as the non-first lines are inden
 ted in the LDIF.
userPassword: eekretsay

"""


class Tree(object):

    def __init__(self):
        global LDIF
        self.f = BytesIO(LDIF)
        d = fromLDIFFile(self.f)
        d.addCallback(self.ldifRead)

    def ldifRead(self, result):
        self.f.close()
        self.db = result


class LDAPServerFactory(ServerFactory):
    protocol = LDAPServer

    def __init__(self, root):
        self.root = root

    def buildProtocol(self, addr):
        proto = self.protocol()
        proto.debug = self.debug
        proto.factory = self
        return proto


if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    from twisted.internet import reactor

    if len(sys.argv) == 2:
        port = int(sys.argv[1])
    else:
        port = 8080
    # First of all, to show logging info in stdout :
    log.startLogging(sys.stderr)
    # We initialize our tree
    tree = Tree()
    # When the LDAP Server protocol wants to manipulate the DIT, it invokes
    # `root = interfaces.IConnectedLDAPEntry(self.factory)` to get the root
    # of the DIT.  The factory that creates the protocol must therefore
    # be adapted to the IConnectedLDAPEntry interface.
    registerAdapter(
        lambda x: x.root,
        LDAPServerFactory,
        IConnectedLDAPEntry)
    factory = LDAPServerFactory(tree.db)
    factory.debug = True
    application = service.Application("ldaptor-server")
    myService = service.IServiceCollection(application)
    serverEndpointStr = "tcp:{0}".format(port)
    e = serverFromString(reactor, serverEndpointStr)
    d = e.listen(factory)
    reactor.run()
```

1.2 User's Guide

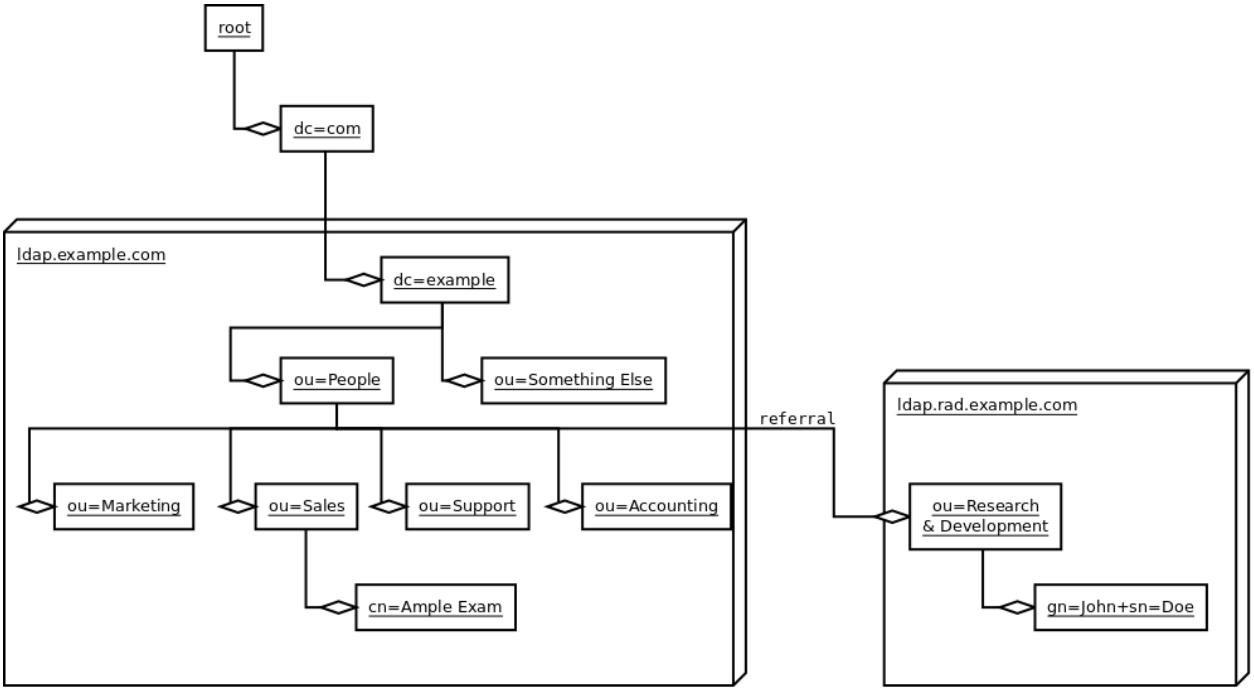
1.2.1 Introduction to LDAP

Foreword

This text is intended as a quick introduction to the interesting bits of the LDAP protocol, and should be useful whether you are managing an LDAP server, programming something using an LDAP library, or writing an LDAP library yourself. I welcome any feedback you might have.

LDAP Presents a Distributed Tree of Information

Probably the nicest way to get a mental model of LDAP information is to think of a tree with elements both in leaf and non-leaf nodes. Parts of the tree may reside at different LDAP servers.



An organization normally uses their DNS domain name as the root entry for their local LDAP tree. For example, `example.com` is free to use `dc=example,dc=com`. The `dc` stands for `domainComponent`. An alternative is to identify the organization via geographical location, as in `o=Example Inc., c=US`, but this is cumbersome as it requires registration to avoid name conflicts. The `o` stands for organization, `c` for country. You will also encounter `ou`, short for organizational unit.

Each node of the tree is called an "LDAP entry", and can contain multiple attributes in the form of attributeType=value pairs, for example `surname=Wiesel`. One attributeType may appear multiple times, in effect having multiple values.

One or more of the attributes are chosen as a Relative Distinguished Name or RDN, and will be used to identify the node based on its parent. This means the RDN must be unique among the children of its parent. Listing all the RDNs, separated by commas, from the node to the root, gives us the Distinguished Name or DN of the entry.

- The RDN of the entry for Jack E. Wiesel is `cn=Jack E. Wiesel`.
- The DN is `cn=Jack E. Wiesel,ou=Sales,ou=People,dc=example,dc=com`.
- The `cn` is short for common name.

The RDN of the entry for John Doe consist of two attributes, `gn=John` and `sn=Doe`, joined with a plus sign to form `gn=John+sn=Doe`. `gn` is short for given name (first name), `sn` for surname (last name).

Objectclasses and Schemas

A special attributeType of `objectClass` lists all the objectclasses the LDAP entry manifests. An object class basically lists what attribute types an entry must have, and what optional attribute types it may have. For example, telephone directory entries must have a name and a telephone number, and may have a fax number and street address. `objectClass` can have multiple values, allowing the same entry to describe e.g. information about a person both for a telephone directory and for UNIX shell login.

An LDAP schema is a part of the configuration of the LDAP server, containing two things: definitions of attribute types and definitions of objectclasses. It is normally stored as ASCII text, but can e.g. be requested from the server over an LDAP connection.

An attribute type definition commonly contains a global identifier for the attribute type (a list of period-separated integers), a list of names for the attribute type, a free-form description and a reference to another attribute type this definition inherits from. It may also contain information about what sort of data the attribute values may contain, how to compare and sort them, how to find substrings in the value, whether the attribute type can have multiple values, etc.

An example attributeType definition:

```
attributetype ( 2.5.4.4 NAME ( 'sn' 'surname' )
DESC 'RFC2256: last (family) name(s) for which the entity is known by'
SUP name )
```

An object class definition also commonly contains a global identifier, name, description and inheritance information. It also commonly lists the attribute types entries having this object class must have, and additional attribute types they may have. An entry cannot have attribute types that are not listed as a `MUST` or `MAY` by one of the entrys object classes or their parents.

An example objectClass definition:

```
objectclass ( 2.5.6.6 NAME 'person' DESC 'RFC2256: a person'
SUP top STRUCTURAL MUST ( sn $ cn ) MAY
( userPassword $ telephoneNumber $ seeAlso $ description ) )
```

There are a lot of pre-existing schemas, standardized in various RFCs. Also, anyone can create their own schemas. The only things you need are access to the LDAP server configuration, and a number reserved for you, which can be achieved by filling a web form.

Object-oriented look at LDAP entries

If you look at LDAP entries from the viewpoint of a programmer accustomed with object oriented programming, you will see a lot of similarities, but also some striking differences.

Writing Things Down: LDIF

There is a standardized way of writing down, in plain text, the contents of LDAP directories, individual entries and even add, delete and modify operations. This format is known as LDIF (LDAP Data Interchange Format) LDAP Data Interchange Format, and it is defined in RFC2849.

The rough format of LDIF is this: there is a paragraph per entry, where paragraphs are separated by blank lines. Each paragraph contains lines in the format keyword:value. Entries start by listing the keyword `dn`, and their DN, and then list all the attributes and values the entry has. Lines starting with space are appended to the previous line. The whole file starts with the keyword version and value 1.

Note: The actual format is more complex, but this tutorial should allow you to read and write normal LDIF files fluently.

A simple LDAP file with two entries:

```
version: 1
dn: cn=Barbara Jensen, ou=Product Development, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Barbara Jensen
cn: Barbara J Jensen
cn: Babs Jensen
sn: Jensen
uid: bjensen
telephonenumber: +1 408 555 1212
description: A big sailing fan.

dn: cn=Bjorn Jensen, ou=Accounting, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Bjorn Jensen
sn: Jensen
telephonenumber: +1 408 555 1212
```

A file containing an entry with a folded attribute value, from RFC 2849:

```
version: 1
dn:cn=Barbara Jensen, ou=Product Development, dc=airius, dc=com
objectclass:top
objectclass:person
objectclass:organizationalPerson
cn:Barbara Jensen
cn:Barbara J Jensen
cn:Babs Jensen
sn:Jensen
uid:bjensen
telephonenumber:+1 408 555 1212
description:Babs is a big sailing fan, and travels extensively in search of perfect␣
↪sailing conditions.
title:Product Manager, Rod and Reel Division
```

Searches and Search Filters

The most common LDAP operation is a search, and LDAP is purposefully designed for environments where searches are many times more common than modify operations. In general, LDAP servers index the entries and can effectively search for matches against a reasonably complex criteria among thousands of entries.

An LDAP search takes the following information as input:

- base DN
- scope (base, one level, subtree)
- filter

- attributes requested

Note: Once again, we are skipping some details for understandability.

Of these, the search filter is clearly the most interesting one. As with LDIF, search filters have a standardized plain text representation, even though they are not transmitted as plain text in the actual protocol.

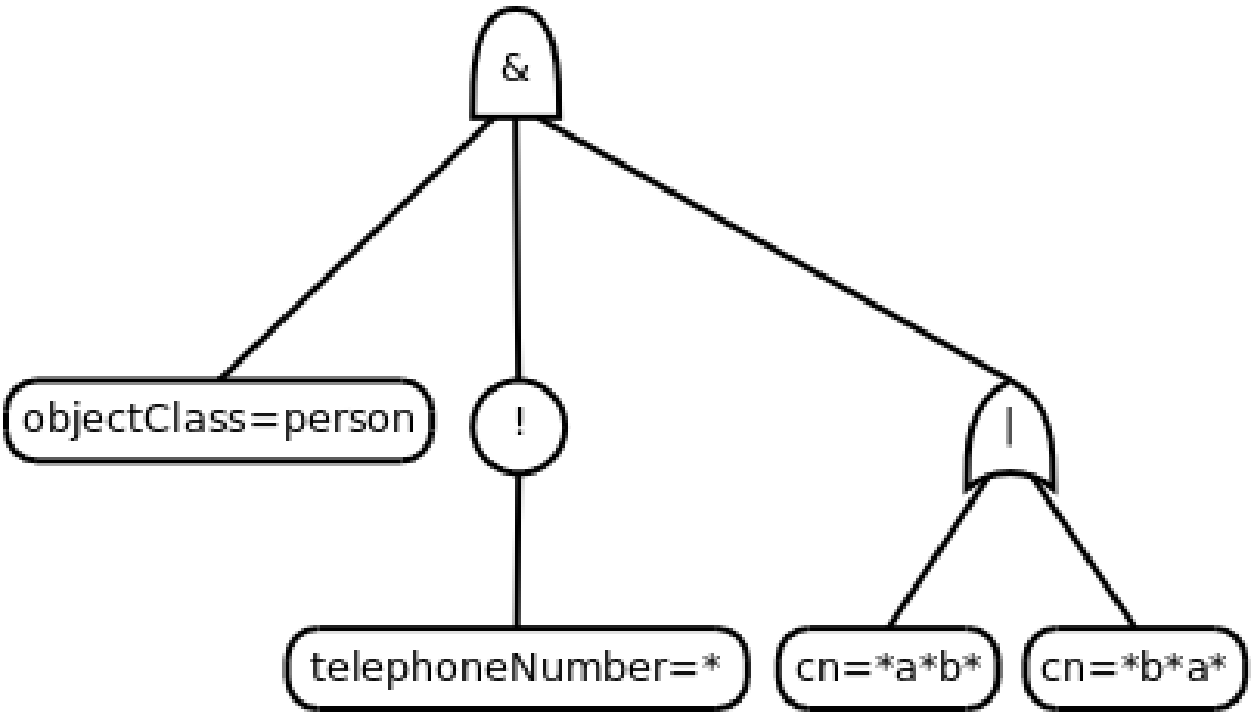
A search filter is basically a combination of tests an entry must fulfill in order to match the filter. They are always written inside parentheses. A simple example would be

```
(cn=John Smith)
```

but the filters can also match against presence, prefix, suffix, substring, rough equality, etc. Multiple matches can be combined freely with and, or and not operators, which are represented by `&`, `|` and `!`, respectively. For example, to match only objects that have objectClass `person`, where the full name contains the letters a and b in either order, and who don't have a telephone number listed, we could use the filter

Note: Yes, once again we are skipping details for understandability. See RFC2254 for more.

```
(&(objectClass=person)(!(telephoneNumber=*))(|(cn=*a*b*)(cn=*b*a*)))
```



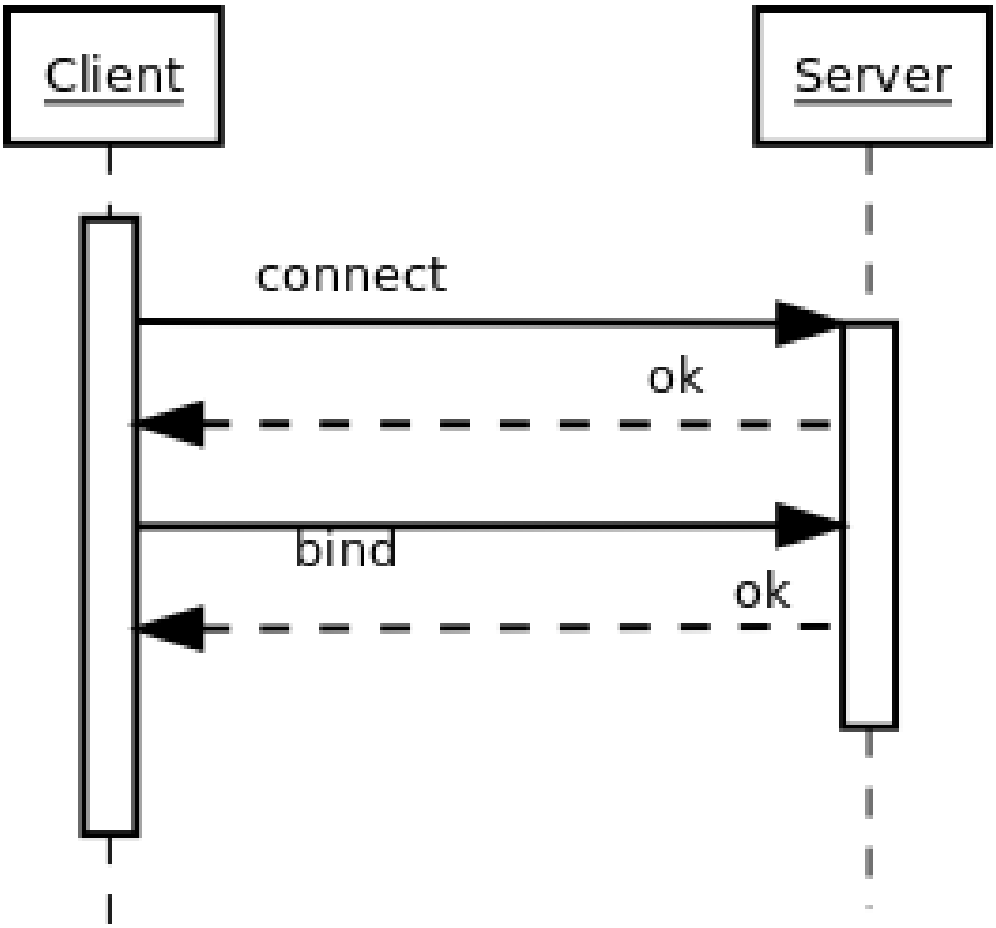
Phases of an LDAP Protocol Chat

An average LDAP protocol chat consists of three stages:

1. Opening the connection
2. Doing one or more searches
3. Closing the connection

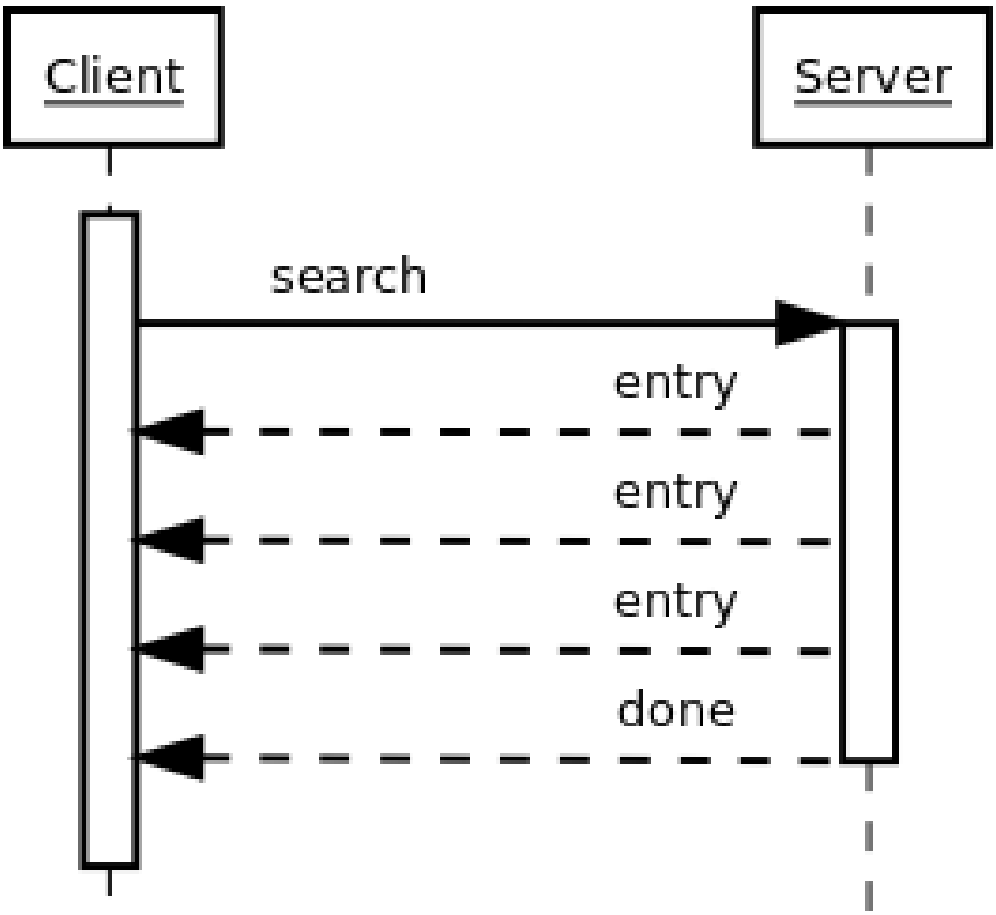
At the first stage, opening a connection, an LDAP client opens a TCP connection to the LDAP server, either as plain text, encrypted by TLS or starting with plaintext and switching to use TLS with STARTTLS.

The client authenticates itself and/or the user, providing any necessary authentication information. This is called binding. Normally, the connection is not really authenticated, but left as anonymous; the bind message is sent with no user or password information.

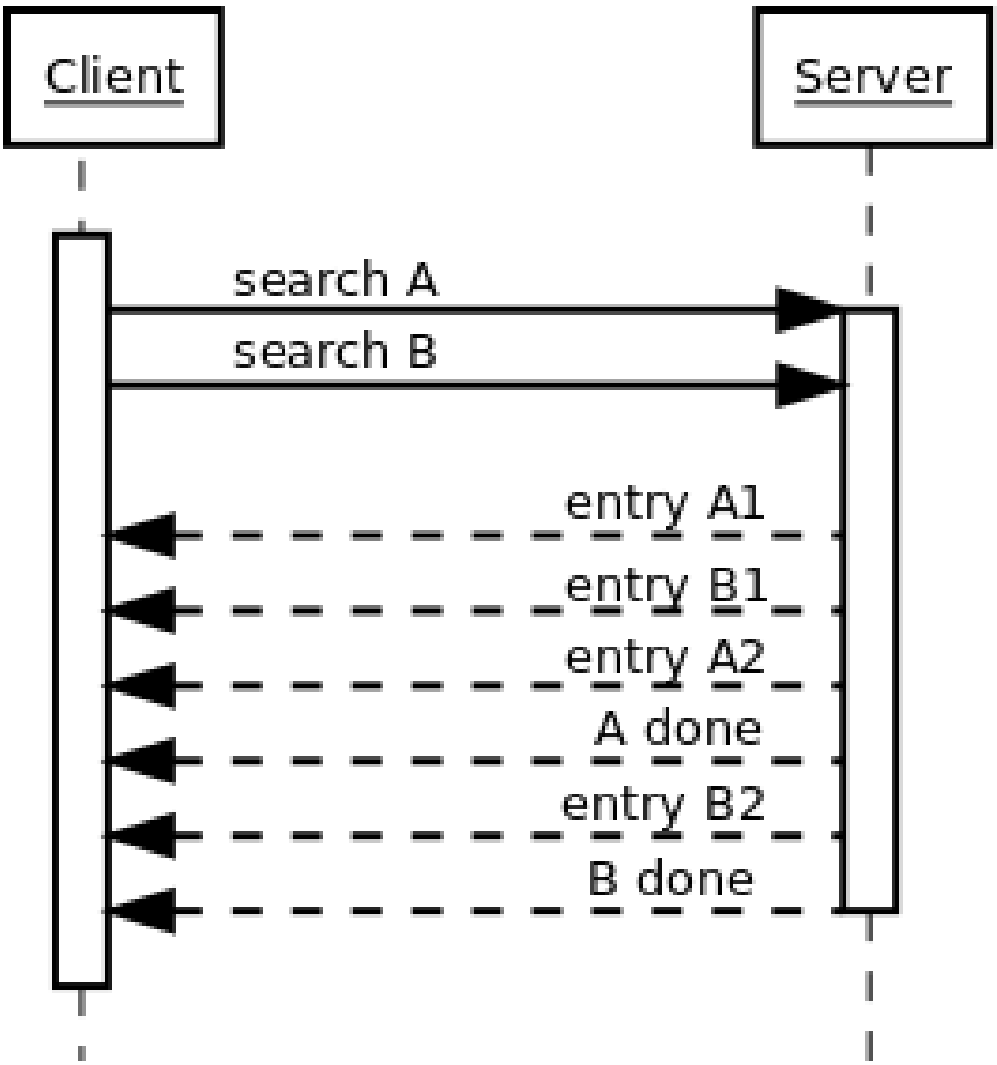


Next, the client sends a search request, containing the base DN for the search, the filter that entries must fulfill to match, and some extra settings discussed above.

The server replies by sending search result entries back, one message per matching entry. If no entry matched or there was an error before the search could even start, the server might not send any entries. Finally, the server sends a message indicating the search is done, and includes information on whether the search was completely successfully, or the error encountered.

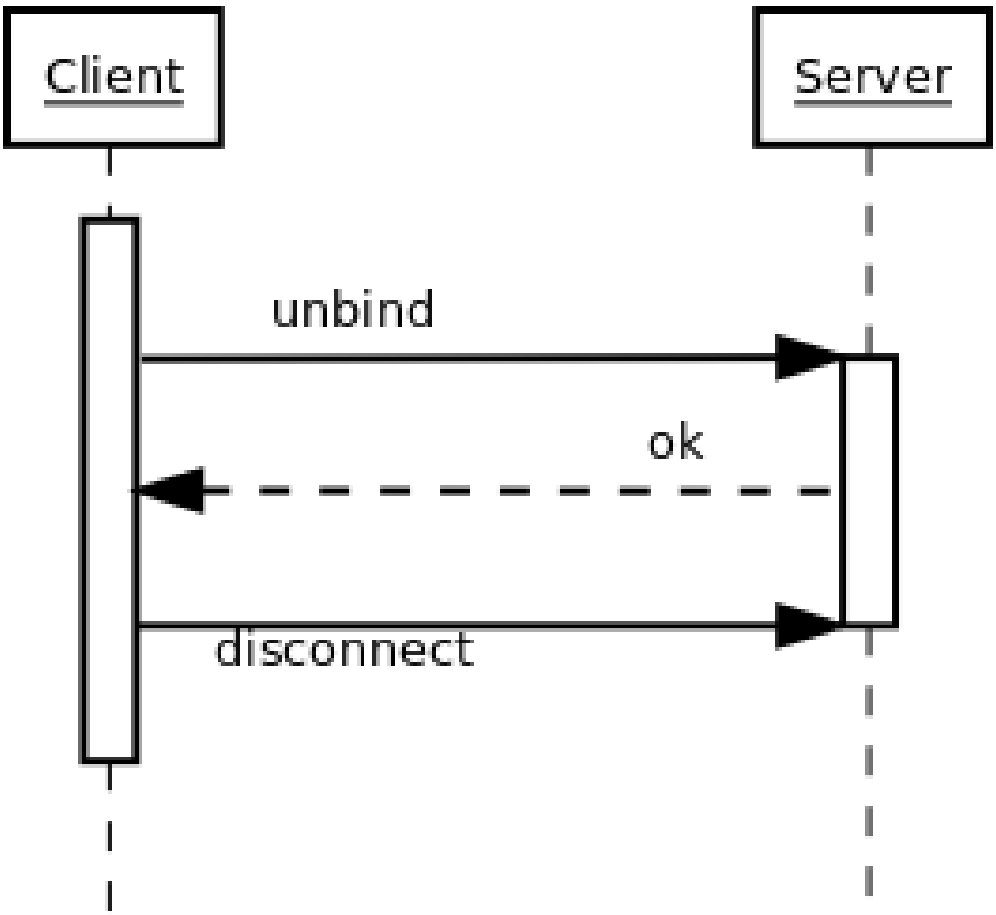


Note that the client could have sent another search request without waiting for the first search to complete. The order of results from the different search, or when they are completed, is in no way guaranteed.



One important detail we have skimmed over so far is how the LDAP client knows what message the server is replying to. Earlier we avoided this topic just by doing only one thing at a time, but now we have two searches getting their result entries interleaved. Clearly, there must be a mechanism to separate which entries belong to which search request. And exactly such a mechanism exists; each message sent by the client contains a number identifying the request, and the server replies by including the same number in the reply. Now, all the client needs to do is remember which numbers are still in use, and not reuse those. It can internally maintain search state based on these numbers, and process result entries based on them. The client can reuse a number when it is known that no more server replies will be sent using that number; for example, the search done message gives this guarantee.

Finally, when the client no longer wants to talk to the server, it sends a message effectively saying "good bye". This message is known as `unbind`. This only means that the state of connection is the same as when connected, before the first `bind`; that is, it un-authenticates the current user. If the client really wants to close the connection, it will then close the TCP socket.

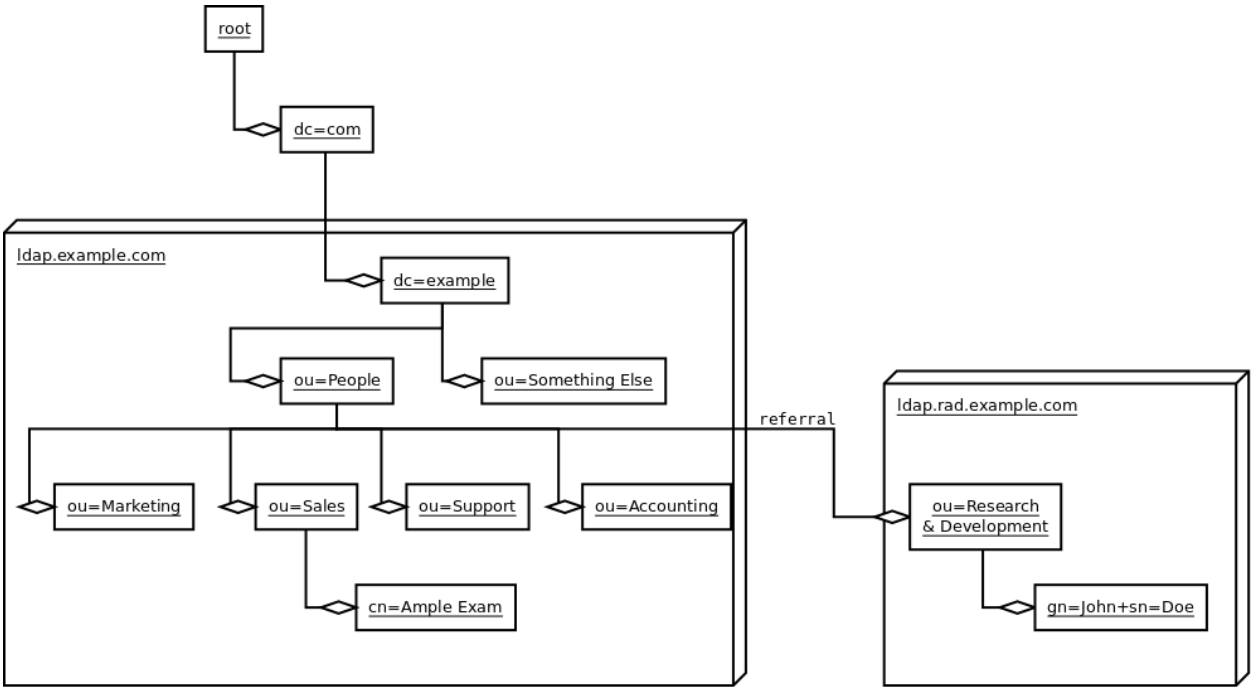


Please understand that these were just examples, and in reality protocol chats are often more complicated. For example, one could connect some other protocol servers, say a web servers, authentication mechanism to actually act as an LDAP client, that tries to bind as the user authenticating himself to the web server, with the password given by the user. If this service had no other interest in the contents of LDAP, it would probably immediately after the bind close the connection. But opening and closing TCP connections repeatedly is slow; it is quite likely the authentication mechanism would be changed to keep a single TCP connection alive, and just do repeated binds over the same connection.

1.2.2 Creating a Simple LDAP Application

An LDAP Primer

Entries in an LDAP directory information tree (DIT) are arranged in a hierarchy.



LDIF is a textual representation of entries in the DIT.

Writing things down, John Doe LDIF:

```
dn: gn=John+sn=Doe,ou=Research & Development,ou=People,dc=example,dc=com
objectClass: addressbookPerson
gn: John
sn: Doe
street: Back alley
postOfficeBox: 123
postalCode: 54321
postalAddress: Backstreet
st: NY
l: New York City
c: US
```

Writing things down, John Smith LDIF:

```
dn: gn=John+sn=Smith,ou=Marketing,ou=People, dc=example,dc=com
objectClass: addressbookPerson
gn: John
sn: Smith
telephoneNumber: 555-1234
facsimileTelephoneNumber: 555-1235
description: This is a description that can span multi
 ple lines as long as the non-first lines are inden
 ted in the LDIF.
```

Twisted

Twisted is an event-driven networking framework written in Python and licensed under the MIT (Expat) License.

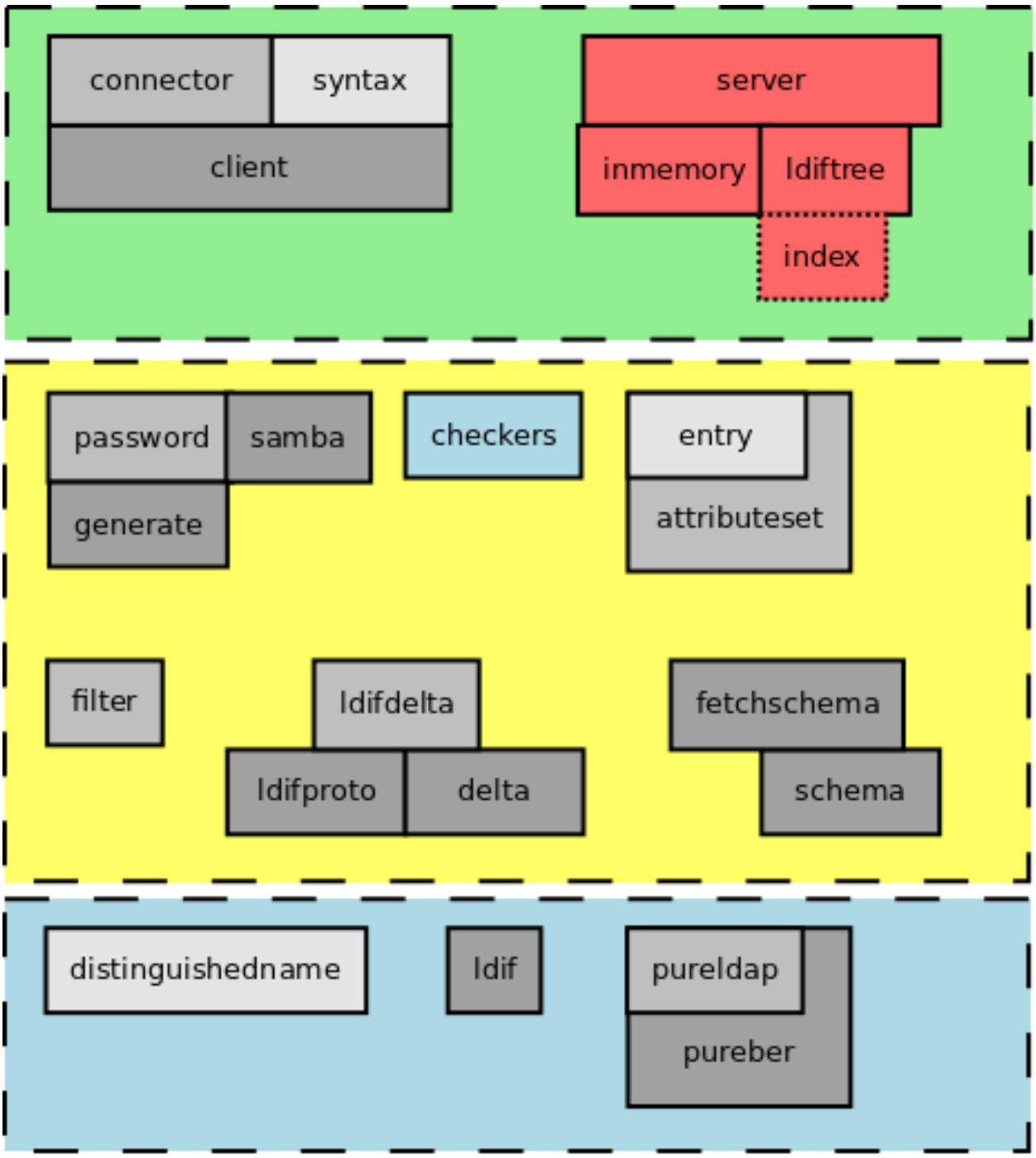
Twisted supports TCP, UDP, SSL/TLS, multicast, Unix sockets, a large number of protocols (including HTTP, NNTP, SSH, IRC, FTP, and others), and much more.

Twisted includes many full-blown applications, such as web, SSH, FTP, DNS and news servers.

Deferreds

- A promise that a function will at some point have a result.
- You can attach callback functions to a Deferred.
- Once it gets a result these callbacks will be called.
- Also allows you to register a callback for an error, with the default behavior of logging the error.
- Standard way to handle all sorts of blocking or delayed operations.

Overview of Ldaptor



Asynchronous LDAP Clients and Servers

Ldaptor is a set of pure-Python LDAP client and server protocols and libraries..

It is licensed under the MIT (Expat) License.

Following Along with the Examples

If you are following along with the interactive examples, you will need an LDAP directory server to which the example client can connect. A script that creates such a server is available in the section *LDAP Server Quick Start*. Copy the script to a file *quickstart_server.py* and run it in another terminal:

```
$ python quickstart_server.py 10389
```

Note: Because of the asynchronous nature of Deferreds, a standard interactive Python shell won't work treat the following examples the way you might expect. That is because the Twisted reator is not running, so connections will never be made and Deferreds will never fire their callback function(s).

If you want to follow along interactively, you can use the following interactive shell that comes with Twisted. It runs a reactor in the background so you can see deferred results:

```
$ python -m twisted.conch.stdio
```

Working with Distinguished Names

```
>>> from ldaptor.protocols.ldap import distinguishedname
>>> dn=distinguishedname.DistinguishedName(
... 'dc=example,dc=com')
>>> dn
DistinguishedName(listOfRDNs=(RelativeDistinguishedName(
attributeTypesAndValues=(LDAPAttributeTypeAndValue(
attributeType='dc', value='example'),)),
RelativeDistinguishedName(attributeTypesAndValues=(
LDAPAttributeTypeAndValue(attributeType='dc', value='com'),))))
>>> str(dn)
'dc=example,dc=com'
```

Connect to a Directory Asynchronously

Ldaptor contains helper classes to simplify connecting to an LDAP DIT.

```
>>> from ldaptor.protocols.ldap.ldapclient import LDAPClient
>>> from twisted.internet import reactor
>>> from twisted.internet.endpoints import clientFromString, connectProtocol
>>> e = clientFromString(reactor, "tcp:host=localhost:port=10389")
>>> e
<twisted.internet.endpoints.TCP4ClientEndpoint at 0xb452e0c>
>>> d = connectProtocol(e, LDAPClient())
>>> d
<Deferred at 0x36755a8 current result: <ldaptor.protocols.ldap.ldapclient.LDAPClient␣
↪instance at 0x36757a0>>
```

Searching

Once connected to the DIT, an LDAP client can search for entries.

```
>>> proto = d.result
>>> proto
<ldaptor.protocols.ldap.ldapclient.LDAPClient instance at 0x40619dac>
>>> from ldaptor.protocols.ldap import ldapsyntax
>>> from ldaptor.protocols.ldap import distinguishedname
>>> dn = distinguishedname.DistinguishedName("dc=example,dc=org")
>>> baseEntry = ldapsyntax.LDAPEntry(client=proto, dn=dn)
>>> d2 = baseEntry.search(filterText='(gn=j*)')
>>> results = d2.result
```

Results

Search results are a list of LDAP entries.

```
>>> results
[LDAPEntry(dn='gn=John+sn=Smith,ou=People,
dc=example,dc=com', attributes={'description': ['Some text.'],
'facsimileTelephoneNumber': ['555-1235'], 'gn': ['John'],
'objectClass': ['addressbookPerson'], 'sn': ['Smith'],
'telephoneNumber': ['555-1234']}), LDAPEntry(dn=
'gn=John+sn=Doe,ou=People,dc=example,dc=com',
attributes={'c': ['US'], 'givenName': ['John'], 'l': ['New York City'],
'objectClass': ['addressbookPerson'], 'postOfficeBox': ['123'],
'postalAddress': ['Backstreet'], 'postalCode': ['54321'],
'sn': ['Doe'], 'st': ['NY'], 'street': ['Back alley']})]
```

Results one-by-one

You can inspect individual results in the result list.

```
>>> results[0]
LDAPEntry(dn=
'gn=John+sn=Smith,ou=People,dc=example,dc=com',
attributes={'description': ['Some text.'],
'facsimileTelephoneNumber': ['555-1235'], 'gn': ['John'],
'objectClass': ['addressbookPerson'], 'sn': ['Smith'],
'telephoneNumber': ['555-1234']})
>>> results[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

LDIF output

Search results can be printed as LDIF output. LDIF output can be used by other LDAP tools.

```
>>> print(results[0])
dn: gn=John+sn=Smith,ou=People,dc=example,dc=com
objectClass: addressbookPerson
description: Some text.
facsimileTelephoneNumber: 555-1235
gn: John
sn: Smith
telephoneNumber: 555-1234
```

Closing the connection

Unlike an HTTP connection, an LDAP connection persists until the client indicates it is done or the server forcibly terminates the connection (e.g. a TCP socket times out).

```
>>> proto.unbind()
```

Access to entry details

LDAP entries have a dictionary-like interface. Attributes are accessed like dictionary keys. The values are always a list of one or more values.

```
>>> smith = results[0]
>>> print(smith.dn)
gn=John+sn=Smith,ou=People,dc=example,dc=com
>>> smith['gn']
['John']
>>>
```

Anatomy of an LDAP entry

LDAP entries can "implement" multiple objectClasses.

All objectClasses can inherit zero, one or many objectClasses, just like programming classes.

All objectClasses have a root class, known as *top*; many object oriented programming languages have a root class, e.g. named *Object*.

All objectClasses are either *STRUCTURAL* or *AUXILIARY*; entries can only implement one *STRUCTURAL* objectClass.

Lastly, objectClasses of an entry can be changed at will; you only need to take care that the entry has all the *MUST* attribute types, and no attribute types outside of the ones that are *MUST* or *MAY*.

Note: Note that e.g. OpenLDAP doesn't implement this.

Attributes of an entry closely match attributes of objects in programming languages; however, LDAP attributes may have multiple values.

Search inputs

An example search filter: `(cn=John Smith)`

A search filter, specifying criteria an entry must fulfill to match.

Scope of the search, either look at the base DN only, only look one level below it, or look at the whole subtree rooted at the base DN.

Size limit of at most how many matching entries to return.

Attributes to return, or none for all attributes the matching entries happen to have.

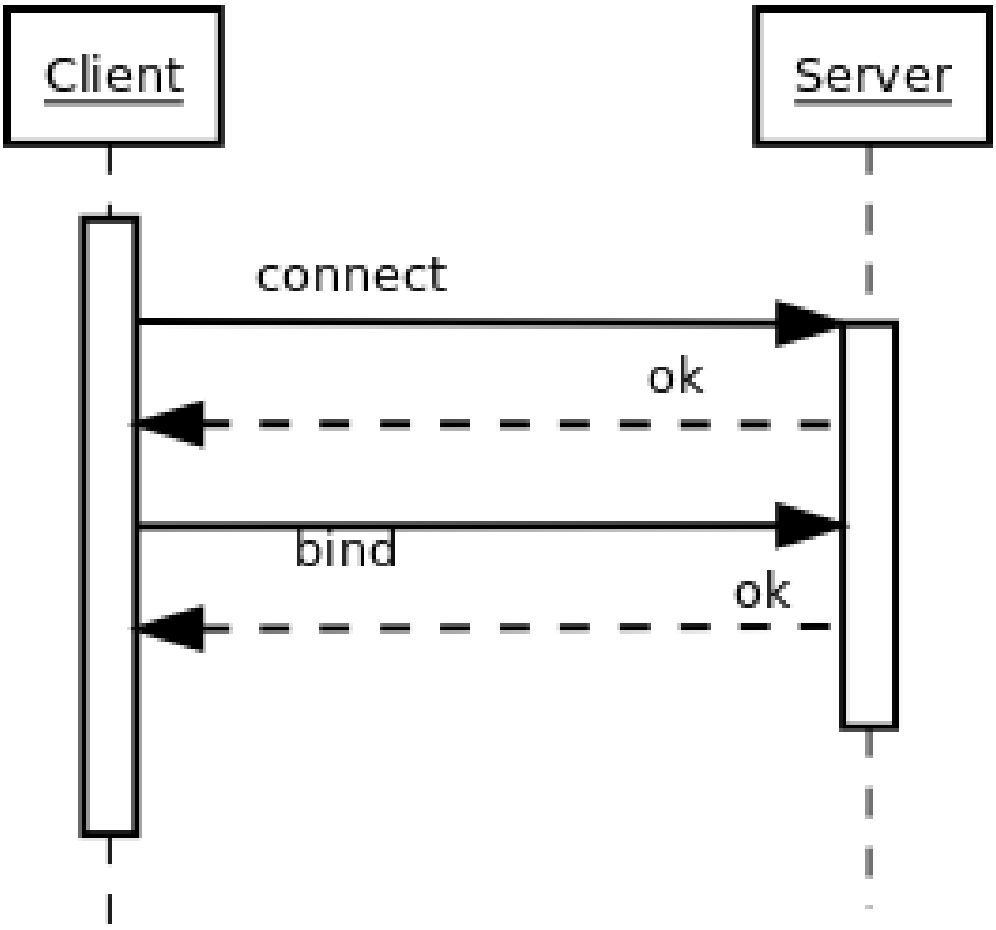
Our first Python program

```
#!/usr/bin/python

from twisted.internet import defer
from twisted.internet.task import react
from twisted.internet.endpoints import clientFromString, connectProtocol
from ldaptor import ldapfilter
from ldaptor.protocols.ldap import ldapsyntax
from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols.ldap.distinguishedname import DistinguishedName

def search(reactor, endpointStr, base_dn):
    e = clientFromString(reactor, endpointStr)
    d = connectToLDAPEndpoint(e, LDAPClient())

    def _doSearch(proto):
        searchFilter = ldapfilter.parseFilter('(gn=j*)')
        baseEntry = ldapsyntax.LDAPEntry(client=proto, dn=base_dn)
        d = baseEntry.search(filterObject=searchFilter)
        return d

    d.addCallback(_doSearch)
    return d

def main(reactor):
    import sys
    from twisted.python import log
    log.startLogging(sys.stderr, setStdout=0)
    dn =  DistinguishedName('dc=example,dc=org')
    d = search(reactor, 'tcp:host=localhost:port=10389', dn)

    def _show(results):
        for item in results:
            print(item)

    d.addCallback(_show)
    d.addErrback(defer.logError)
    d.addBoth(lambda _: reactor.stop())
    return d

if __name__ == '__main__':
    react(main)
```

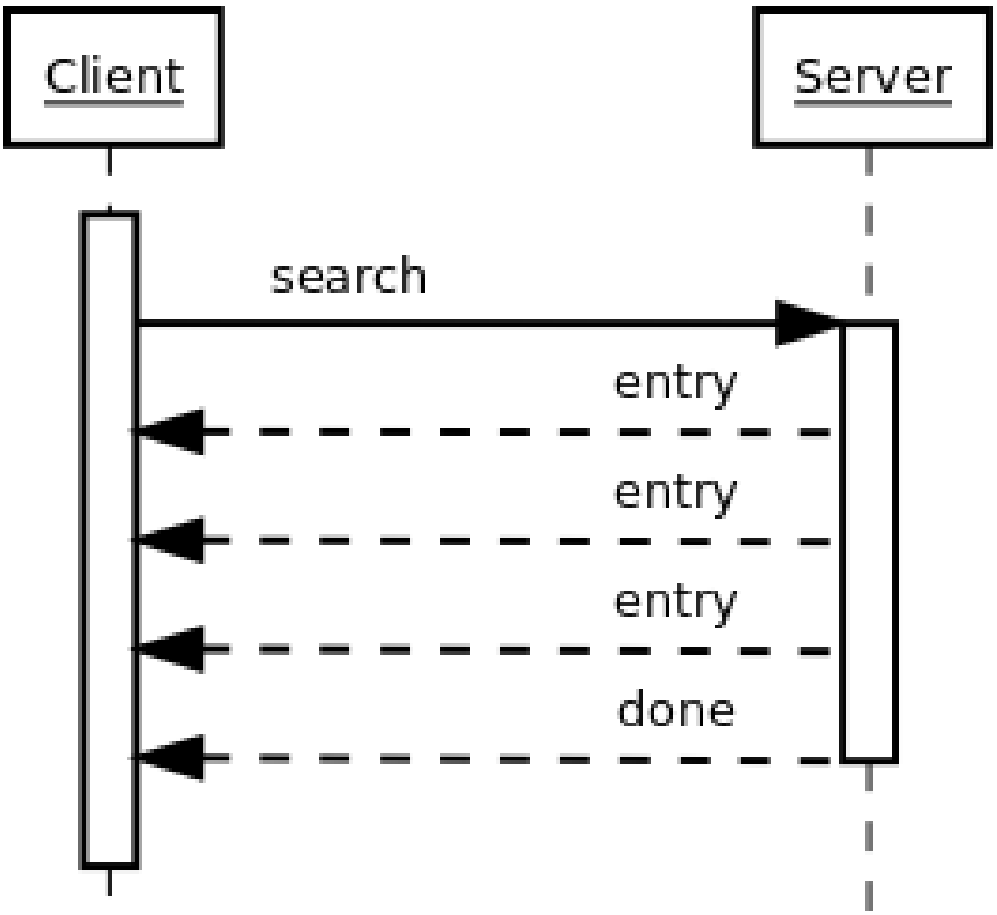
Phases of the protocol chat

- Open and bind
- Search (possibly many times)
- Unbind and close

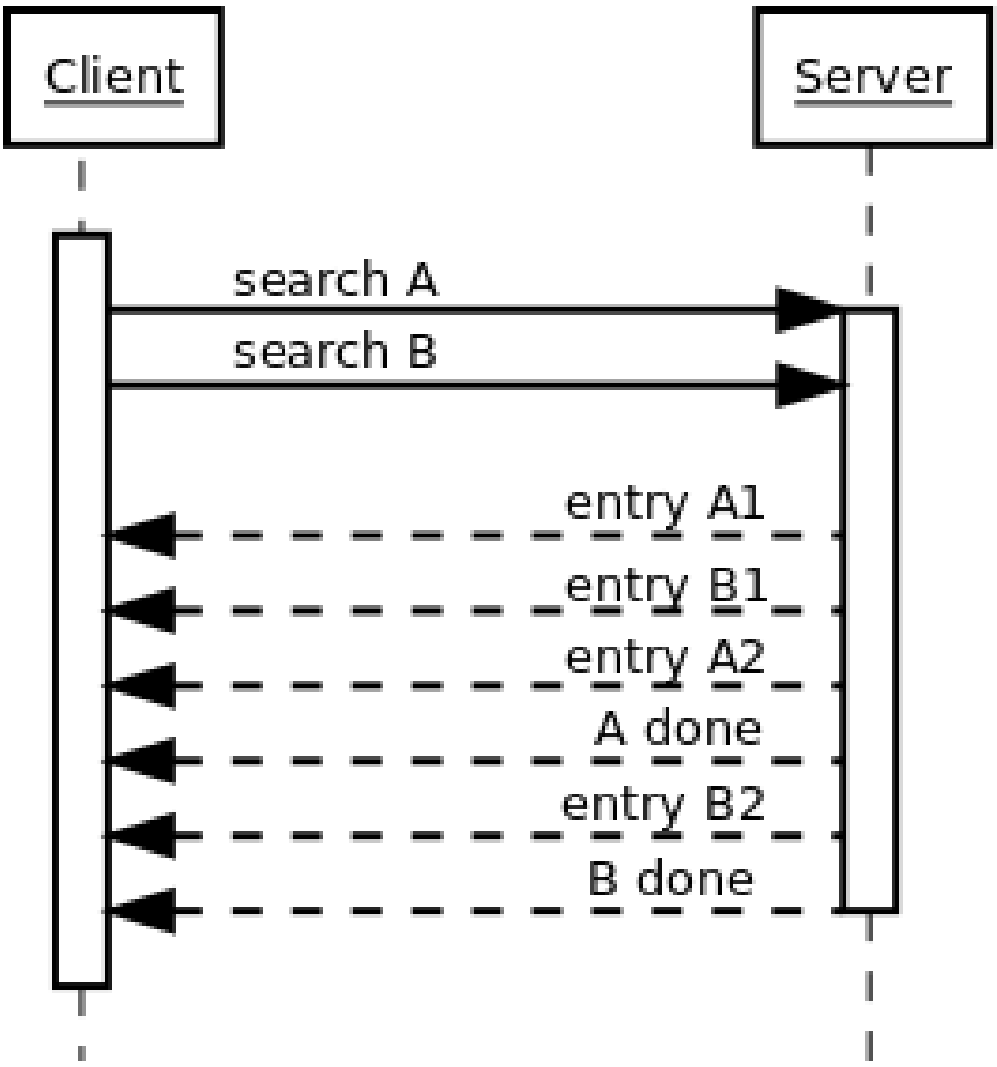
Opening and binding



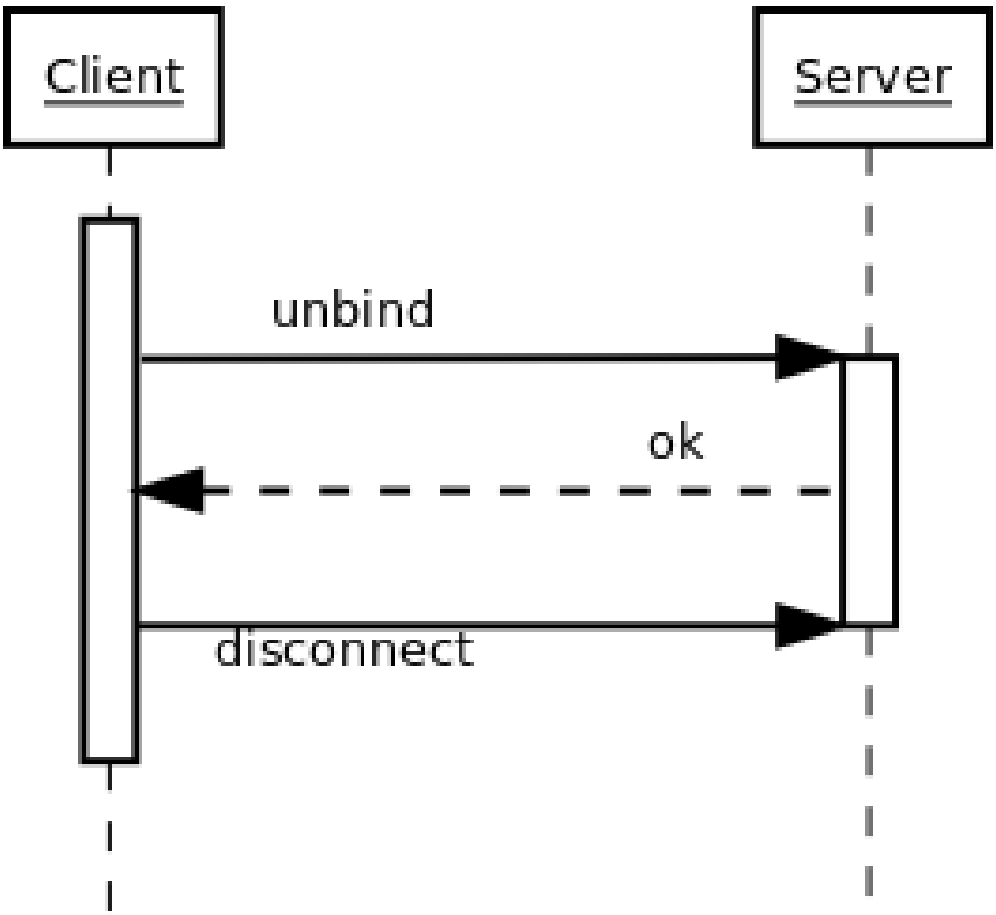
Doing a search



Doing multiple searches



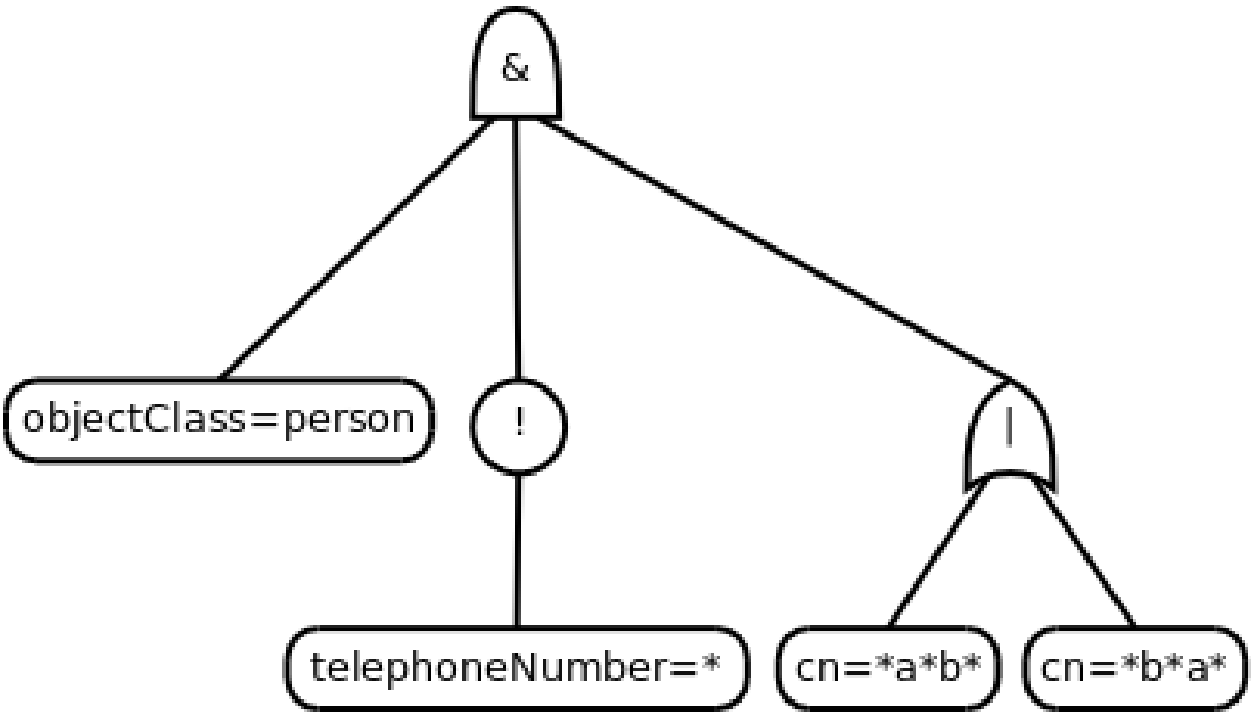
Unbinding and closing



A complex search filter

An example:

```
(&(objectClass=person)
    (!(telephoneNumber=*))
    (|(cn=*a*b*)(cn=*b*a*)))
```



Object classes

1. Special attribute `objectClass` lists all the objectclasses an LDAP entry manifests.
2. **Objectclass defines**
    1. What attributetypes an entry MUST have
    2. What attributetypes an entry MAY have
3. An entry in a phonebook must have a name and a telephone number, and may have a fax number and street address.

Schema

1. A configuration file included in the LDAP server configuration.
2. A combination of attribute type and object class definitions.
3. Stored as plain text
4. Can be requested over an LDAP connection

Attribute type

An example:

```
attributetype ( 2.5.4.4 NAME ( 'sn' 'surname' )
    DESC 'RFC2256: last (family) name(s) for which the entity is known by'
    SUP name )
```

Can also contain:

1. content data type
2. comparison and sort mechanism
3. substring search mechanism
4. whether multiple values are allowed

Object class

An example:

```
objectclass ( 2.5.6.6 NAME 'person'
    DESC 'RFC2256: a person'
    SUP top STRUCTURAL
    MUST ( sn $ cn )
    MAY ( userPassword $ telephoneNumber
    $ seeAlso $ description )
)
```

Creating schemas

1. Anyone can create their own schema
2. Need to be globally unique
3. But try to use already existing ones

Where to go from here?

Install OpenLDAP: http://www.openldap.org/

Install Ldaptor: https://github.com/twisted/ldaptor

Learn Python: http://www.python.org/

Learn Twisted. Write a client application for a simple protocol. Read the HOWTOs: http://twistedmatrix.com/documents/current/core/howto/clients.html

1.2.3 ldaptor API Reference

Subpackages

ldaptor.protocols package

Subpackages

ldaptor.protocols.ldap package

Subpackages

ldaptor.protocols.ldap.autofill package

Submodules

ldaptor.protocols.ldap.autofill.posixAccount module

class ldaptor.protocols.ldap.autofill.posixAccount.**Autofill_posix**(*baseDN*, *freeNumberGetter=<function getFreeNumber>*)

Bases: object

notify(*ldapObject*, *attributeType*)

start(*ldapObject*)

ldaptor.protocols.ldap.autofill.sambaAccount module

class ldaptor.protocols.ldap.autofill.sambaAccount.**Autofill_samba**

Bases: object

notify(*ldapObject*, *attributeType*)

start(*ldapObject*)

ldaptor.protocols.ldap.autofill.sambaSamAccount module

class ldaptor.protocols.ldap.autofill.sambaSamAccount.**Autofill_samba**(*domainSID*, *fixedPrimaryGroupSID=None*)

Bases: object

notify(*ldapObject*, *attributeType*)

start(*ldapObject*)

Module contents

LDAP object field value suggestion and autoupdate mechanism.

exception ldaptor.protocols.ldap.autofill.**AutofillException**
Bases: Exception

exception ldaptor.protocols.ldap.autofill.**ObjectMissingObjectClassException**
Bases: *ldaptor.protocols.ldap.autofill.AutofillException*

The LDAPEntry is missing an objectClass this autofiller needs to operate.

Submodules

ldaptor.protocols.ldap.distinguishedname module

class ldaptor.protocols.ldap.distinguishedname.**DistinguishedName**(*magic=None*, *stringValue=None*, *listOfRDNs=None*)
Bases: ldaptor._encoder.TextStrAlias

LDAP Distinguished Name.

contains(*other*)
Does the tree rooted at DN contain or equal the other DN.

getDomainName()

getText()

listOfRDNs = None

split()

up()

exception ldaptor.protocols.ldap.distinguishedname.**InvalidRelativeDistinguishedName**(*rdn*)
Bases: Exception

Invalid relative distinguished name. It is assumed that passed RDN is of str type: bytes for PY2 and unicode for PY3.

class ldaptor.protocols.ldap.distinguishedname.**LDAPAttributeTypeAndValue**(*stringValue=None*, *attributeType=None*, *value=None*)
Bases: ldaptor._encoder.TextStrAlias

attributeType = None

getText()

value = None

class ldaptor.protocols.ldap.distinguishedname.**RelativeDistinguishedName**(*magic=None*, *stringValue=None*, *attributeTypesAndValues=None*)

Bases: ldaptor._encoder.TextStrAlias

LDAP Relative Distinguished Name.

attributeTypesAndValues = None

count()

getText()

split()

ldaptor.protocols.ldap.distinguishedname.**escape**(*s*)

ldaptor.protocols.ldap.distinguishedname.**unescape**(*s*)

ldaptor.protocols.ldap.fetchschema module

ldaptor.protocols.ldap.fetchschema.**fetch**(*client*, *baseObject*)

ldaptor.protocols.ldap.ldapclient module

LDAP protocol client

class ldaptor.protocols.ldap.ldapclient.**LDAPClient**

Bases: twisted.internet.protocol.Protocol

An LDAP client

berdecoder = <LDAPBERDecoderContext_TopLevel identities={0x10: LDAPMessage} fallback=

bind(*dn=''*, *auth=''*)

@depreciated: Use e.bind(auth).

@todo: Remove this method when there are no callers.

connectionLost(*reason=<twisted.python.failure.Failure twisted.internet.error.ConnectionDone: Connection was closed cleanly.>*)

Called when TCP connection has been lost

connectionMade()

TCP connection has opened

dataReceived(*recd*)

Called whenever data is received.

Use this method to translate to a higher-level message. Usually, some callback will be made upon the receipt of each complete protocol message.

@param data: a string of indeterminate length. Please keep in mind that you will probably need to buffer some data, as partial (or multiple) protocol messages may be received! I recommend that unit tests for protocols call through to this method with differing chunk sizes, down to one byte at a time.

debug = False

handle(*msg*)

send(*op*, *controls=None*)
Send an LDAP operation to the server. @param op: the operation to send @type op: LDAPProtocolRequest @param controls: Any controls to be included in the request. @type controls: LDAPControls @return: the response from server @rtype: Deferred LDAPProtocolResponse

send_multiResponse(*op*, *handler*, **args*, ***kwargs*)
Send an LDAP operation to the server, expecting one or more responses.

If *handler* is provided, it will receive a LDAP response as its first argument. The Deferred returned by this function will never fire.

If *handler* is not provided, the Deferred returned by this function will fire with the final LDAP response.

@param op: the operation to send @type op: LDAPProtocolRequest @param handler: a callable that will be called for each response. It should return a boolean, whether this was the final response. @param args: positional arguments to pass to handler @param kwargs: keyword arguments to pass to handler @return: the result from the first handler as a deferred that completes when the first response has been received @rtype: Deferred LDAPProtocolResponse

send_multiResponse_ex(*op*, *controls=None*, *handler=None*, **args*, ***kwargs*)
Send an LDAP operation to the server, expecting one or more responses.

If *handler* is provided, it will receive a LDAP response *and* response controls as its first 2 arguments. The Deferred returned by this function will never fire.

If *handler* is not provided, the Deferred returned by this function will fire with a tuple of the first LDAP response and any associated response controls.

@param op: the operation to send @type op: LDAPProtocolRequest @param controls: LDAP controls to send with the message. @type controls: LDAPControls @param handler: a callable that will be called for each response. It should return a boolean, whether this was the final response. @param args: positional arguments to pass to handler @param kwargs: keyword arguments to pass to handler @return: the result from the last handler as a deferred that completes when the last response has been received @rtype: Deferred LDAPProtocolResponse

send_noResponse(*op*, *controls=None*)
Send an LDAP operation to the server, with no response expected.

@param op: the operation to send @type op: LDAPProtocolRequest

startTLS(*ctx=None*)
Start Transport Layer Security.

It is the callers responsibility to make sure other things are not happening at the same time.

@todo: server hostname check, see rfc2830 section 3.6. @return: a deferred that will complete when the TLS handshake is complete.

unbind()

unsolicitedNotification(*msg*)

exception ldaptor.protocols.ldap.ldapclient.**LDAPClientConnectionLostException**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

toWire()

exception ldaptor.protocols.ldap.ldapclient.**LDAPStartTLSBusyError**(*onwire*, *message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPOperationsError*

toWire()

exception ldaptor.protocols.ldap.ldapclient.**LDAPStartTLSInvalidResponseName**(*responseName*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

toWire()

ldaptor.protocols.ldap.ldapconnector module

class ldaptor.protocols.ldap.ldapconnector.**LDAPClientCreator**(*reactor*, *protocolClass*, **args*, ***kwargs*)
Bases: twisted.internet.protocol.ClientCreator

connect(*dn*, *overrides=None*, *bindAddress=None*)
Connect to remote host, return Deferred of resulting protocol instance.

connectAnonymously(*dn*, *overrides=None*)
Connect to remote host and bind anonymously, return Deferred of resulting protocol instance.

class ldaptor.protocols.ldap.ldapconnector.**LDAPConnector**(*reactor*, *dn*, *factory*, *overrides=None*, *bindAddress=None*)
Bases: twisted.names.srvconnect.SRVConnector

connect()
Start connection to remote server.

pickServer()
Pick the next server.

ldaptor.protocols.ldap.ldapconnector.**connectToLDAPEndpoint**(*reactor*, *endpointStr*, *clientProtocol*)

ldaptor.protocols.ldap.ldaperrors module

exception ldaptor.protocols.ldap.ldaperrors.**LDAPAdminLimitExceeded**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'adminLimitExceeded'

resultCode = 11

exception ldaptor.protocols.ldap.ldaperrors.**LDAPAffectsMultipleDSAs**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'affectsMultipleDSAs'

resultCode = 71

exception ldaptor.protocols.ldap.ldaperrors.**LDAPAliasDereferencingProblem**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'aliasDereferencingProblem'

resultCode = 36

exception ldaptor.protocols.ldap.ldaperrors.**LDAPAliasProblem**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'aliasProblem'

resultCode = 33

exception ldaptor.protocols.ldap.ldaperrors.**LDAPAttributeOrValueExists**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'attributeOrValueExists'

resultCode = 20

exception ldaptor.protocols.ldap.ldaperrors.**LDAPAuthMethodNotSupported**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'authMethodNotSupported'

resultCode = 7

exception ldaptor.protocols.ldap.ldaperrors.**LDAPBusy**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'busy'

resultCode = 51

exception ldaptor.protocols.ldap.ldaperrors.**LDAPCompareFalse**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'compareFalse'

resultCode = 5

exception ldaptor.protocols.ldap.ldaperrors.**LDAPCompareTrue**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'compareTrue'

resultCode = 6

exception ldaptor.protocols.ldap.ldaperrors.**LDAPConfidentialityRequired**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'confidentialityRequired'

resultCode = 13

exception ldaptor.protocols.ldap.ldaperrors.**LDAPConstraintViolation**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'constraintViolation'

resultCode = 19

exception ldaptor.protocols.ldap.ldaperrors.**LDAPEntryAlreadyExists**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'entryAlreadyExists'

resultCode = 68

exception ldaptor.protocols.ldap.ldaperrors.**LDAPException**(*message=None*)
Bases: Exception, *ldaptor.protocols.ldap.ldaperrors.LDAPResult*

toWire()

class ldaptor.protocols.ldap.ldaperrors.**LDAPExceptionCollection**(*name*, *bases*, *attributes*)

Bases: type

Storage for the LDAP result codes and the corresponding classes.

collection = {0: <class 'ldaptor.protocols.ldap.ldaperrors.Success'>, 1: <class 'lda

classmethod get_instance(*code*, *message*)

Get an instance of the correct exception for this result code.

exception ldaptor.protocols.ldap.ldaperrors.**LDAPInappropriateAuthentication**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'inappropriateAuthentication'

resultCode = 48

exception ldaptor.protocols.ldap.ldaperrors.**LDAPInappropriateMatching**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'inappropriateMatching'

resultCode = 18

exception ldaptor.protocols.ldap.ldaperrors.**LDAPInsufficientAccessRights**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'insufficientAccessRights'

resultCode = 50

exception ldaptor.protocols.ldap.ldaperrors.**LDAPInvalidAttributeSyntax**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'invalidAttributeSyntax'

resultCode = 21

exception ldaptor.protocols.ldap.ldaperrors.**LDAPInvalidCredentials**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'invalidCredentials'

resultCode = 49

exception ldaptor.protocols.ldap.ldaperrors.**LDAPInvalidDNSyntax**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'invalidDNSyntax'

resultCode = 34

exception ldaptor.protocols.ldap.ldaperrors.**LDAPLoopDetect**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'loopDetect'

resultCode = 54

exception ldaptor.protocols.ldap.ldaperrors.**LDAPNamingViolation**(*message=None*)

Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'namingViolation'

resultCode = 64

exception ldaptor.protocols.ldap.ldaperrors.**LDAPNoSuchAttribute**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'noSuchAttribute'

resultCode = 16

exception ldaptor.protocols.ldap.ldaperrors.**LDAPNoSuchObject**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'noSuchObject'

resultCode = 32

exception ldaptor.protocols.ldap.ldaperrors.**LDAPNotAllowedOnNonLeaf**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'notAllowedOnNonLeaf'

resultCode = 66

exception ldaptor.protocols.ldap.ldaperrors.**LDAPNotAllowedOnRDN**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'notAllowedOnRDN'

resultCode = 67

exception ldaptor.protocols.ldap.ldaperrors.**LDAPObjectClassModsProhibited**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'objectClassModsProhibited'

resultCode = 69

exception ldaptor.protocols.ldap.ldaperrors.**LDAPObjectClassViolation**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'objectClassViolation'

resultCode = 65

exception ldaptor.protocols.ldap.ldaperrors.**LDAPOperationsError**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'operationsError'

resultCode = 1

exception ldaptor.protocols.ldap.ldaperrors.**LDAPOther**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'other'

resultCode = 80

exception ldaptor.protocols.ldap.ldaperrors.**LDAPProtocolError**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'protocolError'

resultCode = 2

exception ldaptor.protocols.ldap.ldaperrors.**LDAPReferral**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'referral'

resultCode = 10

class ldaptor.protocols.ldap.ldaperrors.**LDAPResult**
Bases: object

name = None

resultCode = None

exception ldaptor.protocols.ldap.ldaperrors.**LDAPSaslBindInProgress**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'saslBindInProgress'

resultCode = 14

exception ldaptor.protocols.ldap.ldaperrors.**LDAPSizeLimitExceeded**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'sizeLimitExceeded'

resultCode = 4

exception ldaptor.protocols.ldap.ldaperrors.**LDAPStrongAuthRequired**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'strongAuthRequired'

resultCode = 8

exception ldaptor.protocols.ldap.ldaperrors.**LDAPTimeLimitExceeded**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'timeLimitExceeded'

resultCode = 3

exception ldaptor.protocols.ldap.ldaperrors.**LDAPUnavailable**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'unavailable'

resultCode = 52

exception ldaptor.protocols.ldap.ldaperrors.**LDAPUnavailableCriticalExtension**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'unavailableCriticalExtension'

resultCode = 12

exception ldaptor.protocols.ldap.ldaperrors.**LDAPUndefinedAttributeType**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'undefinedAttributeType'

resultCode = 17

exception ldaptor.protocols.ldap.ldaperrors.**LDAPUnknownError**(*resultCode*, *message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

toWire()

exception ldaptor.protocols.ldap.ldaperrors.**LDAPUnwillingToPerform**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

name = b'unwillingToPerform'

resultCode = 53

class ldaptor.protocols.ldap.ldaperrors.**Success**(*msg*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPResult*

name = b'success'

resultCode = 0

ldaptor.protocols.ldap.ldaperrors.**get**(*resultCode*, *errorMessage*)
Get an instance of the correct exception for this resultCode.

ldaptor.protocols.ldap.ldapserver module

LDAP protocol server

class ldaptor.protocols.ldap.ldapserver.**BaseLDAPServer**
Bases: twisted.internet.protocol.Protocol

berdecoder = <LDAPBERDecoderContext_TopLevel identities={0x10: LDAPMessage} fallback=

checkControls(*controls*)

connectionLost(*reason=<twisted.python.failure.Failure twisted.internet.error.ConnectionDone: Connection was closed cleanly.>*)
Called when TCP connection has been lost

connectionMade()
TCP connection has opened

dataReceived(*recd*)
Called whenever data is received.

Use this method to translate to a higher-level message. Usually, some callback will be made upon the receipt of each complete protocol message.

@param data: a string of indeterminate length. Please keep in mind that you will probably need to buffer some data, as partial (or multiple) protocol messages may be received! I recommend that unit tests for protocols call through to this method with differing chunk sizes, down to one byte at a time.

debug = False

failDefault(*resultCode*, *errorMessage*)

handle(*msg*)

handleUnknown(*request*, *controls*, *callback*)

queue(*id*, *op*)

unsolicitedNotification(*msg*)

class ldaptor.protocols.ldap.ldapserver.**LDAPServer**
Bases: *ldaptor.protocols.ldap.ldapserver.BaseLDAPServer*

An LDAP server

boundUser = None

extendedRequest_LDAPPasswordModifyRequest(*data*, *reply*)

fail_LDAPAddRequest
alias of *ldaptor.protocols.pureldap.LDAPAddResponse*

fail_LDAPBindRequest
alias of *ldaptor.protocols.pureldap.LDAPBindResponse*

fail_LDAPCompareRequest
alias of *ldaptor.protocols.pureldap.LDAPCompareResponse*

fail_LDAPDelRequest
alias of *ldaptor.protocols.pureldap.LDAPDelResponse*

fail_LDAPExtendedRequest
alias of *ldaptor.protocols.pureldap.LDAPExtendedResponse*

fail_LDAPModifyDNRequest
alias of *ldaptor.protocols.pureldap.LDAPModifyDNResponse*

fail_LDAPModifyRequest
alias of *ldaptor.protocols.pureldap.LDAPModifyResponse*

fail_LDAPSearchRequest
alias of *ldaptor.protocols.pureldap.LDAPSearchResultDone*

getRootDSE(*request*, *reply*)

handle_LDAPAddRequest(*request*, *controls*, *reply*)

handle_LDAPBindRequest(*request*, *controls*, *reply*)

handle_LDAPCompareRequest(*request*, *controls*, *reply*)

handle_LDAPDelRequest(*request*, *controls*, *reply*)

handle_LDAPExtendedRequest(*request*, *controls*, *reply*)

handle_LDAPModifyDNRequest(*request*, *controls*, *reply*)

handle_LDAPModifyRequest(*request*, *controls*, *reply*)

handle_LDAPSearchRequest(*request*, *controls*, *reply*)

handle_LDAPUnbindRequest(*request*, *controls*, *reply*)

exception ldaptor.protocols.ldap.ldapserver.**LDAPServerConnectionLostException**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPException*

ldaptor.protocols.ldap.ldapsyntax module

Pythonic API for LDAP operations.

exception ldaptor.protocols.ldap.ldapsyntax.**CannotRemoveRDNError**(*key*, *val=None*)
Bases: Exception

The attribute to be removed is the RDN for the object and cannot be removed.

exception ldaptor.protocols.ldap.ldapsyntax.**DNNotPresentError**
Bases: Exception

The requested DN cannot be found by the server.

class ldaptor.protocols.ldap.ldapsyntax.**JournaledLDAPAttributeSet**(*ldapObject*, **a*, ***kw*)
Bases: *ldaptor.attributeset.LDAPAttributeSet*

add(*value*)
Adding key to the attributes with checking if it exists as byte or unicode string

clear()
Remove all elements from this set.

remove(*value*)
Removing key from the attributes with checking if it exists as byte or unicode string

update(*sequence*)
Update a set with the union of itself and others.

ldaptor.protocols.ldap.ldapsyntax.**LDAPEntry**
alias of *ldaptor.protocols.ldap.ldapsyntax.LDAPEntryWithClient*

class ldaptor.protocols.ldap.ldapsyntax.**LDAPEntryWithAutoFill**(**args*, ***kwargs*)
Bases: *ldaptor.protocols.ldap.ldapsyntax.LDAPEntryWithClient*

addAutofiller(*autoFiller*)

journal(*journalOperation*)
Add a Modification into the list of modifications that need to be flushed to the LDAP server.

Normal callers should not use this, they should use the o['foo']=['bar', 'baz'] -style API that enforces schema, handles errors and updates the cached data.

class ldaptor.protocols.ldap.ldapsyntax.**LDAPEntryWithClient**(*client*, *dn*, *attributes={}*, *complete=0*)
Bases: *ldaptor.entry.EditableLDAPEntry*

addChild(*rdn*, *attributes*)

bind(*password*)

buildAttributeSet(*key*, *values*)

commit()

delete()

fetch(**attributes*)

get(**a*, ***kw*)

has_key(**a*, ***kw*)

items()

journal(*journalOperation*)
Add a Modification into the list of modifications that need to be flushed to the LDAP server.

Normal callers should not use this, they should use the o['foo']=['bar', 'baz'] -style API that enforces schema, handles errors and updates the cached data.

keys()

lookup(*dn*)

move(*newDN*)

namingContext()

search(*filterText=None*, *filterObject=None*, *attributes=()*, *scope=None*, *derefAliases=None*, *sizeLimit=0*, *sizeLimitIsNonFatal=False*, *timeLimit=0*, *typesOnly=0*, *callback=None*, *controls=None*, *return_controls=False*)

setPassword(*newPasswd*)
Update the password for the entry with a new password and salt passed as bytes.

setPasswordMaybe_ExtendedOperation(*newPasswd*)
Set the password on this object.

@param newPasswd: A string containing the new password.

@return: A Deferred that will complete when the operation is done.

setPasswordMaybe_Samba(*newPasswd*)
Set the Samba password on this object if it is a sambaSamAccount or sambaAccount.

@param newPasswd: A string containing the new password.

@return: A Deferred that will complete when the operation is done.

setPassword_ExtendedOperation(*newPasswd*)
Set the password on this object.

@param newPasswd: A string containing the new password.

@return: A Deferred that will complete when the operation is done.

setPassword_Samba(*newPasswd*, *style=None*)
Set the Samba password on this object.

@param newPasswd: A string containing the new password.

@param style: one of 'sambaSamAccount', 'sambaAccount' or None. Specifies the style of samba accounts used. None is default and is the same as 'sambaSamAccount'.

@return: A Deferred that will complete when the operation is done.

toWire()

undo()

exception ldaptor.protocols.ldap.ldapsyntax.**MatchNotImplemented**(*op*)
Bases: NotImplementedError

Match type not implemented

exception ldaptor.protocols.ldap.ldapsyntax.**NoContainingNamingContext**
Bases: Exception

The server contains to LDAP naming context that would contain this object.

exception ldaptor.protocols.ldap.ldapsyntax.**ObjectDeletedError**
Bases: *ldaptor.protocols.ldap.ldapsyntax.ObjectInBadStateError*

The LDAP object has already been removed, unable to perform operations on it.

exception ldaptor.protocols.ldap.ldapsyntax.**ObjectDirtyError**
Bases: *ldaptor.protocols.ldap.ldapsyntax.ObjectInBadStateError*

The LDAP object has a journal which needs to be committed or undone before this operation.

exception ldaptor.protocols.ldap.ldapsyntax.**ObjectInBadStateError**
Bases: Exception

The LDAP object in in a bad state.

exception ldaptor.protocols.ldap.ldapsyntax.**PasswordSetAborted**
Bases: Exception

Aborted

exception ldaptor.protocols.ldap.ldapsyntax.**PasswordSetAggregateError**(*errors*)
Bases: Exception

Some of the password plugins failed

ldaptor.protocols.ldap.ldif module

Support for writing a set of directory entries as LDIF. You probably want to use this only indirectly, as in str(LDAPEntry(...)).

TODO support writing modify operations TODO support reading modify operations

TODO implement rest of syntax from RFC2849

ldaptor.protocols.ldap.ldif.**asLDIF**(*dn*, *attributes*)

ldaptor.protocols.ldap.ldif.**attributeAsLDIF**(*attribute*, *value*)

ldaptor.protocols.ldap.ldif.**attributeAsLDIF_base64**(*attribute*, *value*)

ldaptor.protocols.ldap.ldif.**base64_encode**(*s*)

ldaptor.protocols.ldap.ldif.**containsNonprintable**(*s*)

ldaptor.protocols.ldap.ldif.**manyAsLDIF**(*objects*)

ldaptor.protocols.ldap.ldifdelta module

class ldaptor.protocols.ldap.ldifdelta.**LDIFDelta**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIF*

MOD_SPEC_TO_DELTA = {b'add': <class 'ldaptor.delta.Add'>, b'delete': <class 'ldaptor

state_IN_ADD_ENTRY(*line*)

state_IN_DELETE(*line*)

state_IN_MOD_SPEC(*line*)

state_WAIT_FOR_CHANGETYPE(*line*)

state_WAIT_FOR_DN(*line*)

state_WAIT_FOR_MOD_SPEC(*line*)

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaAddMissingAttributesError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

Add operation needs to have at least one attribute type and value.

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaDeleteHasJunkAfterChangeTypeError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

Delete operation takes no attribute types or values.

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaMissingChangeTypeError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF delta entry has no changetype.

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaModificationDifferentAttributeTypeError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

The attribute type for the change is not the as in the mod-spec header line.

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaModificationMissingEndDashError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF delta modification has no ending dash.

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaUnknownChangeTypeError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF delta entry has an unknown changetype.

exception ldaptor.protocols.ldap.ldifdelta.**LDIFDeltaUnknownModificationError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF delta modification has unknown mod-spec.

ldaptor.protocols.ldap.ldifdelta.**fromLDIFFile**(*f*)
Read LDIF data from a file.

ldaptor.protocols.ldap.ldifprotocol module

class ldaptor.protocols.ldap.ldifprotocol.**LDIF**
Bases: twisted.protocols.basic.LineReceiver, object

connectionLost(*reason=<twisted.python.failure.Failure twisted.internet.error.ConnectionDone: Connection was closed cleanly.>*)
Called when the connection is shut down.

Clear any circular references here, and any external references to this Protocol. The connection has been closed.

@type reason: L{twisted.python.failure.Failure}

data = None

delimiter = b'\n'

dn = None

gotEntry(*obj*)

lastLine = None

lineReceived(*line*)
Override this for when each line is received.

@param line: The line which was received with the delimiter removed. @type line: C{bytes}

logicalLineReceived(*line*)

mode = b'HEADER'

parseValue(*val*)

state_HEADER(*line*)

state_IN_ENTRY(*line*)

state_WAIT_FOR_DN(*line*)

version = None

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFEntryStartsWithNonDNError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF entry starts with a non-DN line

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFEntryStartsWithSpaceError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

Invalid LDIF value format

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFLineWithoutSemicolonError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF line without semicolon seen

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFParseError**
Bases: Exception

Error parsing LDIF

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFTruncatedError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF appears to be truncated

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFUnsupportedVersionError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

LDIF version not supported

exception ldaptor.protocols.ldap.ldifprotocol.**LDIFVersionNotANumberError**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIFParseError*

Non-numeric LDIF version number

ldaptor.protocols.ldap.proxy module

LDAP protocol proxy server

class ldaptor.protocols.ldap.proxy.**Proxy**(*config*)
Bases: *ldaptor.protocols.ldap.ldapserver.BaseLDAPServer*

client = None

connectionLost(*reason*)
Called when TCP connection has been lost

connectionMade()
TCP connection has opened

handleUnknown(*request*, *controls*, *reply*)

handle_LDAPUnbindRequest(*request*, *controls*, *reply*)

protocol
alias of *ldaptor.protocols.ldap.ldapclient.LDAPClient*

unbound = False

waitingConnect = []

ldaptor.protocols.ldap.svcbindproxy module

class ldaptor.protocols.ldap.svcbindproxy.**ServiceBindingProxy**(*services=None*, *fallback=None*, **a*, ***kw*)

Bases: *ldaptor.protocols.ldap.proxy.Proxy*

An LDAP proxy that handles non-anonymous bind requests specially.

BindRequests are intercepted and authentication is attempted against each configured service. This authentication is performed against a separate LDAP entry, found by searching for entries with

- objectClass: serviceSecurityObject
- owner: the DN of the original bind attempt
- cn: the service name.

starting at the identity-base as configured in the config file.

Finally, if the authentication does not succeed against any of the configured services, the proxy can fallback to passing the bind request to the real server.

fail_LDAPBindRequest
alias of *ldaptor.protocols.pureldap.LDAPBindResponse*

fallback = False

handle_LDAPBindRequest(*request*, *controls*, *reply*)

services = []

timestamp()

Module contents

LDAP protocol logic

Submodules

ldaptor.protocols.pureber module

Pure, simple, BER encoding and decoding

class ldaptor.protocols.pureber.**BERBase**(*tag=None*)
Bases: ldaptor._encoder.WireStrAlias

identification()

tag = None

toWire()

class ldaptor.protocols.pureber.**BERBoolean**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERBase*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 1

toWire()

class ldaptor.protocols.pureber.**BERDecoderContext**(*fallback=None*, *inherit=None*)
Bases: object

Identities = {1: <class 'ldaptor.protocols.pureber.BERBoolean'>, 2: <class 'ldaptor.

inherit()

lookup_id(*id*)

class ldaptor.protocols.pureber.**BEREnumerated**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERInteger*

tag = 10

exception ldaptor.protocols.pureber.**BERException**
Bases: Exception

exception ldaptor.protocols.pureber.**BERExceptionInsufficientData**
Bases: Exception

class ldaptor.protocols.pureber.**BERInteger**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERBase*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 2

toWire()

value = None

class ldaptor.protocols.pureber.**BERNull**(*tag=None*)
Bases: *ldaptor.protocols.pureber.BERBase*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 5

toWire()

class ldaptor.protocols.pureber.**BEROctetString**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERBase*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 4

toWire()

value = None

class ldaptor.protocols.pureber.**BERSequence**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERStructured*, collections.UserList

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 16

toWire()

class ldaptor.protocols.pureber.**BERSequenceOf**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERSequence*

class ldaptor.protocols.pureber.**BERSet**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERSequence*

tag = 17

class ldaptor.protocols.pureber.**BERStructured**(*tag=None*)
Bases: *ldaptor.protocols.pureber.BERBase*

identification()

exception ldaptor.protocols.pureber.**UnknownBERTag**(*tag*, *context*)
Bases: Exception

ldaptor.protocols.pureber.**ber2int**(*e*, *signed=True*)

ldaptor.protocols.pureber.**berDecodeLength**(*m*, *offset=0*)
Return a tuple of (length, lengthLength). m must be atleast one byte long.

ldaptor.protocols.pureber.**berDecodeMultiple**(*content*, *berdecoder*) → [objects]
Decodes everything in content and returns a list of decoded objects.

All of content will be decoded, and content must contain complete BER objects.

ldaptor.protocols.pureber.**berDecodeObject**(*context*, *bytes*)
berobject may be None.

ldaptor.protocols.pureber.**int2ber**(*i*, *signed=True*)

ldaptor.protocols.pureber.**int2berlen**(*i*)

ldaptor.protocols.pureber.**need**(*buf*, *n*)

ldaptor.protocols.pureldap module

LDAP protocol message conversion; no application logic here.

class ldaptor.protocols.pureldap.**LDAPAbandonRequest**(*value=None*, *id=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureldap.LDAPInteger*

needs_answer = 0

tag = 80

toWire()

class ldaptor.protocols.pureldap.**LDAPAddRequest**(*entry=None*, *attributes=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 72

toWire()

class ldaptor.protocols.pureldap.**LDAPAddResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPResult*

tag = 73

class ldaptor.protocols.pureldap.**LDAPAssertionValue**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BEROctetString*

class ldaptor.protocols.pureldap.**LDAPAttributeDescription**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

class ldaptor.protocols.pureldap.**LDAPAttributeValue**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

class ldaptor.protocols.pureldap.**LDAPAttributeValueAssertion**(*attributeDesc=None*, *assertionValue=None*, *tag=None*, *escaper=<function escape>*)

Bases: *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

toWire()

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {64: <class 'ldaptor.protocols.pureldap.LDAPBindRequest'>, 65: <class '

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_BindResponse**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {135: <class 'ldaptor.protocols.pureldap.LDAPBindResponse_serverSaslCred

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_Compare**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {16: <class 'ldaptor.protocols.pureldap.LDAPAttributeValueAssertion'>}

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_Filter**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {128: <class 'ldaptor.protocols.pureldap.LDAPFilter_and'>, 129: <class

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_Filter_substrings**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {128: <class 'ldaptor.protocols.pureldap.LDAPFilter_substrings_initial'>

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPBindRequest**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {128: <class 'ldaptor.protocols.pureber.BEROctetString'>, 131: <class '

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPControls**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {16: <class 'ldaptor.protocols.pureldap.LDAPControl'>}

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPExtendedRequest**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {128: <class 'ldaptor.protocols.pureber.BEROctetString'>, 129: <class '

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPExtendedResponse**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {138: <class 'ldaptor.protocols.pureldap.LDAPResponseName'>, 139: <class

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPMessage**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {83: <class 'ldaptor.protocols.pureldap.LDAPSearchResultReference'>, 128

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPPasswordModifyRequest**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {128: <class 'ldaptor.protocols.pureldap.LDAPPasswordModifyRequest_userI

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_LDAPSearchResultReference**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {4: <class 'ldaptor.protocols.pureldap.LDAPString'>}

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_MatchingRuleAssertion**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {129: <class 'ldaptor.protocols.pureldap.LDAPMatchingRuleAssertion_matchi

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_ModifyDNRequest**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {128: <class 'ldaptor.protocols.pureldap.LDAPModifyDNResponse_newSuperio

class ldaptor.protocols.pureldap.**LDAPBERDecoderContext_TopLevel**(*fallback=None*, *inherit=None*)

Bases: *ldaptor.protocols.pureber.BERDecoderContext*

Identities = {16: <class 'ldaptor.protocols.pureldap.LDAPMessage'>}

class ldaptor.protocols.pureldap.**LDAPBindRequest**(*version=None*, *dn=None*, *auth=None*, *tag=None*, *sasl=False*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 64

toWire()

class ldaptor.protocols.pureldap.**LDAPBindResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPResult*

errorMessage = None

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

matchedDN = None

referral = None

resultCode = None

serverSaslCreds = None

tag = 65

class ldaptor.protocols.pureldap.**LDAPBindResponse_serverSaslCreds**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

tag = 135

class ldaptor.protocols.pureldap.**LDAPCompareRequest**(*entry*, *ava*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

ava = None

entry = None

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 78

toWire()

class ldaptor.protocols.pureldap.**LDAPCompareResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPResult*

tag = 79

class ldaptor.protocols.pureldap.**LDAPControl**(*controlType*, *criticality=None*, *controlValue=None*, *id=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BERSequence*

controlValue = None

criticality = None

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

toWire()

class ldaptor.protocols.pureldap.**LDAPControls**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 128

class ldaptor.protocols.pureldap.**LDAPDelRequest**(*value=None*, *entry=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureldap.LDAPString*

tag = 74

toWire()

class ldaptor.protocols.pureldap.**LDAPDelResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPResult*

tag = 75

class ldaptor.protocols.pureldap.**LDAPExtendedRequest**(*requestName=None*, *requestValue=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

requestName = None

requestValue = None

tag = 87

toWire()

class ldaptor.protocols.pureldap.**LDAPExtendedResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *responseName=None*, *response=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPResult*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

response = None

responseName = None

tag = 88

toWire()

class ldaptor.protocols.pureldap.**LDAPFilter**(*tag=None*)
Bases: *ldaptor.protocols.pureber.BERStructured*

class ldaptor.protocols.pureldap.**LDAPFilterSet**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERSet*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

class ldaptor.protocols.pureldap.**LDAPFilter_and**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPFilterSet*

asText()

tag = 128

class ldaptor.protocols.pureldap.**LDAPFilter_approxMatch**(*attributeDesc=None*, *assertionValue=None*, *tag=None*, *escaper=<function escape>*)

Bases: *ldaptor.protocols.pureldap.LDAPAttributeValueAssertion*

asText()

tag = 136

class ldaptor.protocols.pureldap.**LDAPFilter_equalityMatch**(*attributeDesc=None*, *assertionValue=None*, *tag=None*, *escaper=<function escape>*)

Bases: *ldaptor.protocols.pureldap.LDAPAttributeValueAssertion*

asText()

tag = 131

class ldaptor.protocols.pureldap.**LDAPFilter_extensibleMatch**(*matchingRule=None*, *type=None*, *matchValue=None*, *dnAttributes=None*, *tag=None*, *escaper=<function escape>*)

Bases: *ldaptor.protocols.pureldap.LDAPMatchingRuleAssertion*

asText()

tag = 137

class ldaptor.protocols.pureldap.**LDAPFilter_greaterOrEqual**(*attributeDesc=None*, *assertionValue=None*, *tag=None*, *escaper=<function escape>*)

Bases: *ldaptor.protocols.pureldap.LDAPAttributeValueAssertion*

asText()

tag = 133

class ldaptor.protocols.pureldap.**LDAPFilter_lessOrEqual**(*attributeDesc=None*, *assertionValue=None*, *tag=None*, *escaper=<function escape>*)

Bases: *ldaptor.protocols.pureldap.LDAPAttributeValueAssertion*

asText()

tag = 134

class ldaptor.protocols.pureldap.**LDAPFilter_not**(*value*, *tag=130*)

Bases: *ldaptor.protocols.pureldap.LDAPFilter*

asText()

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 130

toWire()

class ldaptor.protocols.pureldap.**LDAPFilter_or**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPFilterSet*

asText()

tag = 129

class ldaptor.protocols.pureldap.**LDAPFilter_present**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureldap.LDAPAttributeDescription*

asText()

tag = 135

class ldaptor.protocols.pureldap.**LDAPFilter_substrings**(*type=None*, *substrings=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERSequence*

asText()

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 132

toWire()

class ldaptor.protocols.pureldap.**LDAPFilter_substrings_any**(**args*, ***kwargs*)
Bases: *ldaptor.protocols.pureldap.LDAPString*

asText()

tag = 129

class ldaptor.protocols.pureldap.**LDAPFilter_substrings_final**(**args*, ***kwargs*)
Bases: *ldaptor.protocols.pureldap.LDAPString*

asText()

tag = 130

class ldaptor.protocols.pureldap.**LDAPFilter_substrings_initial**(**args*, ***kwargs*)
Bases: *ldaptor.protocols.pureldap.LDAPString*

asText()

tag = 128

class ldaptor.protocols.pureldap.**LDAPInteger**(*value=None*, *tag=None*)
Bases: *ldaptor.protocols.pureber.BERInteger*

class ldaptor.protocols.pureldap.**LDAPMatchingRuleAssertion**(*matchingRule=None*, *type=None*, *matchValue=None*, *dnAttributes=None*, *tag=None*, *escaper=<function escape>*)
Bases: *ldaptor.protocols.pureber.BERSequence*

dnAttributes = None

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

matchValue = None

matchingRule = None

toWire()

type = None

class ldaptor.protocols.pureldap.**LDAPMatchingRuleAssertion_dnAttributes**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BERBoolean*

tag = 132

class ldaptor.protocols.pureldap.**LDAPMatchingRuleAssertion_matchValue**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPAssertionValue*

tag = 131

class ldaptor.protocols.pureldap.**LDAPMatchingRuleAssertion_matchingRule**(**args*, ***kwargs*)

Bases: *ldaptor.protocols.pureldap.LDAPMatchingRuleId*

tag = 129

class ldaptor.protocols.pureldap.**LDAPMatchingRuleAssertion_type**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPAttributeDescription*

tag = 130

class ldaptor.protocols.pureldap.**LDAPMatchingRuleId**(**args*, ***kwargs*)

Bases: *ldaptor.protocols.pureldap.LDAPString*

class ldaptor.protocols.pureldap.**LDAPMessage**(*value=None*, *controls=None*, *id=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BERSequence*

To encode this object in order to be sent over the network use the toWire() method.

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

id = None

toWire()

This is the wire/encoded representation.

value = None

class ldaptor.protocols.pureldap.**LDAPModifyDNRequest**(*entry*, *newrdn*, *deleteoldrdn*, *newSuperior=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

deleteoldrdn = None

entry = None

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

newSuperior = None

newrdn = None

tag = 76

toWire()

class ldaptor.protocols.pureldap.**LDAPModifyDNResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPResult*

tag = 77

class ldaptor.protocols.pureldap.**LDAPModifyDNResponse_newSuperior**(**args*, ***kwargs*)

Bases: *ldaptor.protocols.pureldap.LDAPString*

tag = 128

class ldaptor.protocols.pureldap.**LDAPModifyRequest**(*object=None*, *modification=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

modification = None

object = None

tag = 70

toWire()

class ldaptor.protocols.pureldap.**LDAPModifyResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPResult*

tag = 71

class ldaptor.protocols.pureldap.**LDAPOID**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

class ldaptor.protocols.pureldap.**LDAPPasswordModifyRequest**(*requestName=None*, *userIdentity=None*, *oldPasswd=None*, *newPasswd=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPExtendedRequest*

oid = b'1.3.6.1.4.1.4203.1.11.1'

class ldaptor.protocols.pureldap.**LDAPPasswordModifyRequest_newPasswd**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPPasswordModifyRequest_passwd*

tag = 130

class ldaptor.protocols.pureldap.**LDAPPasswordModifyRequest_oldPasswd**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPPasswordModifyRequest_passwd*

tag = 129

class ldaptor.protocols.pureldap.**LDAPPasswordModifyRequest_passwd**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

class ldaptor.protocols.pureldap.**LDAPPasswordModifyRequest_userIdentity**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

tag = 128

class ldaptor.protocols.pureldap.**LDAPProtocolOp**

Bases: object

toWire()

class ldaptor.protocols.pureldap.**LDAPProtocolRequest**

Bases: *ldaptor.protocols.pureldap.LDAPProtocolOp*

needs_answer = 1

class ldaptor.protocols.pureldap.**LDAPProtocolResponse**

Bases: *ldaptor.protocols.pureldap.LDAPProtocolOp*

class ldaptor.protocols.pureldap.**LDAPReferral**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BERSequence*

tag = 131

class ldaptor.protocols.pureldap.**LDAPResponse**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

tag = 139

class ldaptor.protocols.pureldap.**LDAPResponseName**(*value=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPOID*

tag = 138

class ldaptor.protocols.pureldap.**LDAPResult**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolResponse*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

toWire()

class ldaptor.protocols.pureldap.**LDAPSearchRequest**(*baseObject=None*, *scope=None*, *derefAliases=None*, *sizeLimit=None*, *timeLimit=None*, *typesOnly=None*, *filter=None*, *attributes=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERSequence*

attributes = []

baseObject = ''

derefAliases = 0

filter = LDAPFilter_present(value='objectClass')

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

scope = 2

sizeLimit = 0

tag = 67

timeLimit = 0

toWire()

typesOnly = 0

class ldaptor.protocols.pureldap.**LDAPSearchResultDone**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPResult*

tag = 69

class ldaptor.protocols.pureldap.**LDAPSearchResultEntry**(*objectName*, *attributes*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolResponse*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 68

toWire()

class ldaptor.protocols.pureldap.**LDAPSearchResultReference**(*uris=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPProtocolResponse*, *ldaptor.protocols.pureber.BERSequence*

classmethod fromBER(*tag*, *content*, *berdecoder=None*)

tag = 83

toWire()

class ldaptor.protocols.pureldap.**LDAPStartTLSRequest**(*requestName=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPExtendedRequest*

Request to start Transport Layer Security. See RFC 2830 for details.

oid = b'1.3.6.1.4.1.1466.20037'

class ldaptor.protocols.pureldap.**LDAPStartTLSResponse**(*resultCode=None*, *matchedDN=None*, *errorMessage=None*, *referral=None*, *serverSaslCreds=None*, *responseName=None*, *response=None*, *tag=None*)

Bases: *ldaptor.protocols.pureldap.LDAPExtendedResponse*

Response to start Transport Layer Security. See RFC 4511 section 4.14.2 for details.

oid = b'1.3.6.1.4.1.1466.20037'

class ldaptor.protocols.pureldap.**LDAPString**(**args*, ***kwargs*)

Bases: *ldaptor.protocols.pureber.BEROctetString*

class ldaptor.protocols.pureldap.**LDAPUnbindRequest**(**args*, ***kwargs*)
Bases: *ldaptor.protocols.pureldap.LDAPProtocolRequest*, *ldaptor.protocols.pureber.BERNull*

needs_answer = 0

tag = 66

toWire()

ldaptor.protocols.pureldap.**alloc_ldap_message_id**()

ldaptor.protocols.pureldap.**binary_escape**(*s*)

ldaptor.protocols.pureldap.**escape**(*s*)

ldaptor.protocols.pureldap.**smart_escape**(*s*, *threshold=0.3*)

Module contents

ldaptor.samba package

Submodules

ldaptor.samba.smbpassword module

ldaptor.samba.smbpassword.**lmhash**(*password=b''*)
Generates lanman password hash for a given password.

Note that the author thinks LanMan hashes should be banished from the face of the earth.

ldaptor.samba.smbpassword.**lmhash_locked**(*password=b''*)
Generates a lanman password hash that matches no password.

Note that the author thinks LanMan hashes should be banished from the face of the earth.

ldaptor.samba.smbpassword.**nthash**(*password=b''*)
Generates nt md4 password hash for a given password.

Module contents

Submodules

ldaptor.attributeset module

class ldaptor.attributeset.**LDAPAttributeSet**(*key*, **a*, ***kw*)
Bases: set

add(*key*)
Adding key to the attributes with checking if it exists as byte or unicode string

copy()
Return a shallow copy of a set.

remove(*key*)
Removing key from the attributes with checking if it exists as byte or unicode string

ldaptor.checkers module

class ldaptor.checkers.**LDAPBindingChecker**(*cfg*)
Bases: object

The avatarID returned is an LDAPEntry.

credentialInterfaces = (<InterfaceClass twisted.cred.credentials.IUsernamePassword>,)

requestAvatarId(*credentials*)

ldaptor.checkers.**makeFilter**(*name*, *template=None*)

ldaptor.config module

class ldaptor.config.**LDAPConfig**(*baseDN=None*, *serviceLocationOverrides=None*, *identityBaseDN=None*, *identitySearch=None*)
Bases: object

baseDN = None

copy(***kw*)

getBaseDN()

getIdentityBaseDN()

getIdentitySearch(*name*)

getServiceLocationOverrides()

identityBaseDN = None

identitySearch = None

exception ldaptor.config.**MissingBaseDNError**
Bases: Exception

Configuration must specify a base DN

ldaptor.config.**loadConfig**(*configFiles=None*, *reload=False*)
Load configuration file.

ldaptor.config.**useLMhash**()
Read configuration file if necessary and return whether to use LanMan hashes or not.

ldaptor.delta module

Changes to the content of one single LDAP entry.

(This means these do not belong here: adding or deleting of entries, changing of location in tree)

class ldaptor.delta.**Add**(*key*, **a*, ***kw*)
Bases: *ldaptor.delta.Modification*

asLDIF()

patch(*entry*)

class ldaptor.delta.**AddOp**(*entry*)
Bases: *ldaptor.delta.Operation*

asLDIF()

patch(*root*)
Find the correct entry in IConnectedLDAPEntry and patch it.

@param root: IConnectedLDAPEntry that is at the root of the subtree the patch applies to.

@returns: Deferred with None or failure.

class ldaptor.delta.**Delete**(*key*, **a*, ***kw*)
Bases: *ldaptor.delta.Modification*

asLDIF()

patch(*entry*)

class ldaptor.delta.**DeleteOp**(*dn*)
Bases: *ldaptor.delta.Operation*

asLDIF()

patch(*root*)
Find the correct entry in IConnectedLDAPEntry and patch it.

@param root: IConnectedLDAPEntry that is at the root of the subtree the patch applies to.

@returns: Deferred with None or failure.

class ldaptor.delta.**Modification**(*key*, **a*, ***kw*)
Bases: *ldaptor.attributeset.LDAPAttributeSet*

asLDAP()

patch(*entry*)

class ldaptor.delta.**ModifyOp**(*dn*, *modifications=[]*)
Bases: *ldaptor.delta.Operation*

asLDAP()

asLDIF()

classmethod fromLDAP(*request*)

patch(*root*)
Find the correct entry in IConnectedLDAPEntry and patch it.

@param root: IConnectedLDAPEntry that is at the root of the subtree the patch applies to.

@returns: Deferred with None or failure.

class ldaptor.delta.**Operation**
Bases: object

patch(*root*)
Find the correct entry in IConnectedLDAPEntry and patch it.

@param root: IConnectedLDAPEntry that is at the root of the subtree the patch applies to.

@returns: Deferred with None or failure.

class ldaptor.delta.**Replace**(*key*, **a*, ***kw*)
Bases: *ldaptor.delta.Modification*

asLDIF()

patch(*entry*)

ldaptor.dns module

DNS-related utilities.

ldaptor.dns.**aton**(*ip*)

ldaptor.dns.**aton_numbits**(*num*)

ldaptor.dns.**aton_octets**(*ip*)

ldaptor.dns.**netmaskToNumbits**(*netmask*)

ldaptor.dns.**ntoa**(*n*)

ldaptor.dns.**ptrSoaName**(*ip*, *netmask*)
Convert an IP address and netmask to a CIDR delegation -style zone name.

ldaptor.entry module

class ldaptor.entry.**BaseLDAPEntry**(*dn*, *attributes={}*)
Bases: ldaptor._encoder.WireStrAlias

bind(*password*)

buildAttributeSet(*key*, *values*)

diff(*other*)
Compute differences between this and another LDAP entry.

@param other: An LDAPEntry to compare to.

@return: None if equal, otherwise a ModifyOp that would make this entry look like other.

dn = None

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

getLDIF()

hasMember(*dn*)

has_key(*key*)

items()

keys()

toWire()

class ldaptor.entry.**EditableLDAPEntry**(*dn*, *attributes={}*)
Bases: *ldaptor.entry.BaseLDAPEntry*

commit()

delete()

move(*newDN*)

setPassword(*newPasswd*, *salt=None*)
Update the password for the entry with a new password and salt passed as bytes.

undo()

ldaptor.entry.**sshaDigest**(*passphrase*, *salt=None*)
Return the salted SHA for *passphrase* which is passed as bytes.

ldaptor.entryhelpers module

class ldaptor.entryhelpers.**DiffTreeMixin**
Bases: object

diffTree(*other*, *result=None*)

class ldaptor.entryhelpers.**MatchMixin**
Bases: object

match(*filter*)

class ldaptor.entryhelpers.**SearchByTreeWalkingMixin**
Bases: object

search(*filterText=None*, *filterObject=None*, *attributes=()*, *scope=None*, *derefAliases=None*, *sizeLimit=0*, *timeLimit=0*, *typesOnly=0*, *callback=None*)

class ldaptor.entryhelpers.**SubtreeFromChildrenMixin**
Bases: object

subtree(*callback=None*)

ldaptor.entryhelpers.**safelower**(*s*)
As string.lower(), but return *s* if something goes wrong.

ldaptor.generate_password module

exception ldaptor.generate_password.**PwgenException**
Bases: Exception

class ldaptor.generate_password.**ReadPassword**(*deferred*, *count=1*)
Bases: twisted.internet.protocol.ProcessProtocol

errReceived(*data*)
Some data was received from stderr.

outReceived(*data*)
Some data was received from stdout.

processEnded(*reason*)
Called when the child process exits and all file descriptors associated with it have been closed.

@type reason: L{twisted.python.failure.Failure}

ldaptor.generate_password.**generate**(*reactor*, *n=1*)

ldaptor.inmemory module

class ldaptor.inmemory.**InMemoryLDIFProtocol**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIF*

Receive LDIF data and gather results into an ReadOnlyInMemoryLDAPEntry.

You can override lookupFailed and addFailed to provide smarter error handling. They are called as Deferred errbacks; returning the reason causes error to pass onward and abort the whole operation. Returning None from lookupFailed skips that entry, but continues loading.

When the full LDIF data has been read, the completed Deferred will trigger.

addFailed(*reason*, *entry*)

connectionLost(*reason*)
Called when the connection is shut down.

Clear any circular references here, and any external references to this Protocol. The connection has been closed.

@type reason: L{twisted.python.failure.Failure}

gotEntry(*entry*)

lookupFailed(*reason*, *entry*)

exception ldaptor.inmemory.**LDAPCannotRemoveRootError**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPNamingViolation*

Cannot remove root of LDAP tree

class ldaptor.inmemory.**ReadOnlyInMemoryLDAPEntry**(**a*, ***kw*)
Bases: *ldaptor.entry.EditableLDAPEntry*, *ldaptor.entryhelpers.DiffTreeMixin*, *ldaptor.entryhelpers.SubtreeFromChildrenMixin*, *ldaptor.entryhelpers.MatchMixin*, *ldaptor.entryhelpers.SearchByTreeWalkingMixin*

addChild(*rdn*, *attributes*)
TODO ugly API. Returns the created entry.

children(*callback=None*)

commit()

delete()

deleteChild(*rdn*)

fetch(**attributes*)

lookup(*dn*)

move(*newDN*)

parent()

ldaptor.inmemory.**fromLDIFFile**(*f*)
Read LDIF data from a file.

ldaptor.interfaces module

ldaptor.ldapfilter module

exception ldaptor.ldapfilter.**InvalidLDAPFilter**(*msg*, *loc*, *text*)
Bases: Exception

ldaptor.ldapfilter.**parseExtensible**(*attr*, *s*)

ldaptor.ldapfilter.**parseFilter**(*s*)
Converting source string to pureldap.LDAPFilter

Source string is converted to unicode for Python 3 as pyparsing cannot parse Python 3 byte strings with the rules declared in this module.

ldaptor.ldapfilter.**parseMaybeSubstring**(*attrType*, *s*)

ldaptor.ldiftree module

Manage LDAP data as a tree of LDIF files.

exception ldaptor.ldiftree.**LDAPCannotRemoveRootError**(*message=None*)
Bases: *ldaptor.protocols.ldap.ldaperrors.LDAPNamingViolation*

Cannot remove root of LDAP tree

class ldaptor.ldiftree.**LDIFTreeEntry**(*path*, *dn=None*, **a*, ***kw*)
Bases: *ldaptor.entry.EditableLDAPEntry*, *ldaptor.entryhelpers.DiffTreeMixin*, *ldaptor.entryhelpers.SubtreeFromChildrenMixin*, *ldaptor.entryhelpers.MatchMixin*, *ldaptor.entryhelpers.SearchByTreeWalkingMixin*

addChild(*rdn*, *attributes*)

children(*callback=None*)

commit()

delete()

deleteChild(*rdn*)

lookup(*dn*)

move(*newDN*)

parent()

exception ldaptor.ldiftree.**LDIFTreeEntryContainsMultipleEntries**
Bases: Exception

LDIFTree entry contains multiple LDIF entries.

exception ldaptor.ldiftree.**LDIFTreeEntryContainsNoEntries**
Bases: Exception

LDIFTree entry does not contain a valid LDIF entry.

exception ldaptor.ldiftree.**LDIFTreeNoSuchObject**
Bases: Exception

LDIFTree does not contain such entry.

class ldaptor.ldiftree.**StoreParsedLDIF**
Bases: *ldaptor.protocols.ldap.ldifprotocol.LDIF*

connectionLost(*reason*)
Called when the connection is shut down.

Clear any circular references here, and any external references to this Protocol. The connection has been closed.

@type reason: L{twisted.python.failure.Failure}

gotEntry(*obj*)

ldaptor.ldiftree.**get**(*path*, *dn*)

ldaptor.ldiftree.**put**(*path*, *entry*)

ldaptor.numberalloc module

Find an available uidNumber/gidNumber/other similar number.

class ldaptor.numberalloc.**freeNumberGuesser**(*makeAGuess*, *min=None*, *max=None*)
Bases: object

startGuessing()

ldaptor.numberalloc.**getFreeNumber**(*ldapObject*, *numberType*, *min=None*, *max=None*)

class ldaptor.numberalloc.**ldapGuesser**(*ldapObject*, *numberType*)
Bases: object

guess(*num*)

ldaptor.schema module

class ldaptor.schema.**ASN1ParserThingie**
Bases: object

class ldaptor.schema.**AttributeTypeDescription**(*text*)
Bases: *ldaptor.schema.ASN1ParserThingie*, ldaptor._encoder.WireStrAlias

ASN Syntax:

```
AttributeTypeDescription = "(" whsp
        numericoid whsp                 ; AttributeType identifier
        [ "NAME" qdescrs ]              ; name used in AttributeType
        [ "DESC" qdstring ]             ; description
        [ "OBSOLETE" whsp ]
        [ "SUP" woid ]                  ; derived from this other AttributeType
        [ "EQUALITY" woid               ; Matching Rule name
        [ "ORDERING" woid               ; Matching Rule name
        [ "SUBSTR" woid ]               ; Matching Rule name
        [ "SYNTAX" whsp noidlen whsp ] ; see section 4.3
        [ "SINGLE-VALUE" whsp ]         ; default multi-valued
        [ "COLLECTIVE" whsp ]           ; default not collective
        [ "NO-USER-MODIFICATION" whsp ]; default user modifiable
        [ "USAGE" whsp AttributeUsage ]; default userApplications
        whsp ")"

AttributeUsage =
        "userApplications"     /
        "directoryOperation"   /
        "distributedOperation" / ; DSA-shared
        "dSAOperation"          ; DSA-specific, value depends on server

noidlen = numericoid [ "{" len "}" ]

len     = numericstring
```

toWire()

class ldaptor.schema.**MatchingRuleDescription**(*text*)
Bases: *ldaptor.schema.ASN1ParserThingie*, ldaptor._encoder.WireStrAlias

ASN Syntax:

```
MatchingRuleDescription = "(" whsp
        numericoid whsp  ; MatchingRule identifier
        [ "NAME" qdescrs ]
        [ "DESC" qdstring ]
        [ "OBSOLETE" whsp ]
        "SYNTAX" numericoid
        whsp ")"
```

toWire()

class ldaptor.schema.**ObjectClassDescription**(*text*)

Bases: *ldaptor.schema.ASN1ParserThingie*, ldaptor._encoder.WireStrAlias

ASN Syntax:

```
d               = "0" / "1" / "2" / "3" / "4" /
                  "5" / "6" / "7" / "8" / "9"

numericstring   = 1*d

numericoid      = numericstring *( "." numericstring )

space           = 1*" "

whsp            = [ space ]

descr           = keystring

qdescr          = whsp "'" descr "'" whsp

qdescrlist      = [ qdescr *( qdescr ) ]

; object descriptors used as schema element names
qdescrs         = qdescr / ( whsp "(" qdescrlist ")" whsp )

dstring         = 1*utf8

qdstring        = whsp "'" dstring "'" whsp

descr           = keystring

oid             = descr / numericoid

woid            = whsp oid whsp

; set of oids of either form
oids            = woid / ( "(" oidlist ")" )

ObjectClassDescription = "(" whsp
        numericoid whsp      ; ObjectClass identifier
        [ "NAME" qdescrs ]
        [ "DESC" qdstring ]
        [ "OBSOLETE" whsp ]
        [ "SUP" oids ]       ; Superior ObjectClasses
        [ ( "ABSTRACT" / "STRUCTURAL" / "AUXILIARY" ) whsp ]
                             ; default structural
        [ "MUST" oids ]      ; AttributeTypes
        [ "MAY" oids ]       ; AttributeTypes
```

(continues on next page)

(continued from previous page)

```
        whsp ")"
```

toWire()

class ldaptor.schema.**SyntaxDescription**(*text*)
Bases: *ldaptor.schema.ASN1ParserThingie*, ldaptor._encoder.WireStrAlias

ASN Syntax:

```
SyntaxDescription = "(" whsp
        numericoid whsp
        [ "DESC" qdstring ]
        whsp ")"
```

toWire()

ldaptor.schema.**extractWord**(*text*)

ldaptor.schema.**peekWord**(*text*)

ldaptor.testutil module

Utilities for writing Twistedy unit tests and debugging.

class ldaptor.testutil.**FakeTransport**(*proto*)
Bases: object

loseConnection()

class ldaptor.testutil.**LDAPClientTestDriver**(**responses*)
Bases: object

A test driver that looks somewhat like a real LDAPClient.

Pass in a list of lists of LDAPProtocolResponses. For each sent LDAP message, the first item of said list is iterated through, and all the items are sent as responses to the callback. The sent LDAP messages are stored in self.sent, so you can assert that the sent messages are what they are supposed to be.

It is also possible to include a Failure instance instead of a list of LDAPProtocolResponses which will cause the errback to be called with the failure.

assertNothingSent()

assertSent(**shouldBeSent*)

connectionLost(*reason=None*)
Called when TCP connection has been lost

connectionMade()
TCP connection has opened

fakeUnbindResponse = 'fake-unbind-by-LDAPClientTestDriver'

send(*op*)

send_multiResponse(*op*, *handler*, **args*, ***kwargs*)

send_multiResponse_(*op*, *controls*, *return_controls*, *handler*, **args*, ***kwargs*)

send_multiResponse_ex(*op*, *controls*, *handler*, **args*, ***kwargs*)

send_noResponse(*op*)

unbind()

ldaptor.testutil.**calltrace**()
Print out all function calls. For debug use only.

ldaptor.testutil.**createServer**(*proto*, **responses*, ***kw*)
Create an LDAP server for testing. :param proto: The server protocol factory (e.g. *ProxyBase*). :param responses: The responses to initialize the *LDAPClientTestDrive*. :param proto_args: Optional mapping passed as keyword args to protocol factory.

ldaptor.testutil.**mustRaise**(*dummy*)

ldaptor.usage module

Command line argument/options available to various ldaptor tools.

class ldaptor.usage.**Options**
Bases: twisted.python.usage.Options

optParameters = ()

postOptions()
I am called after the options are parsed.

Override this method in your subclass to do something after the options have been parsed and assigned, like validate that all options are sane.

class ldaptor.usage.**Options_base**
Bases: *ldaptor.usage.Options_base_optional*

postOptions_base()

class ldaptor.usage.**Options_base_optional**
Bases: object

optParameters = (('base', None, None, 'LDAP base dn'),)

class ldaptor.usage.**Options_bind**
Bases: object

optParameters = (('binddn', None, None, 'use Distinguished Name to bind to the directo

postOptions_bind_auth_fd_numeric()

class ldaptor.usage.**Options_bind_mandatory**
Bases: *ldaptor.usage.Options_bind*

postOptions_bind_mandatory()

class ldaptor.usage.**Options_scope**
Bases: object

optParameters = (('scope', None, 'sub', 'LDAP search scope (one of base, one, sub)'),)

postOptions_scope()

class ldaptor.usage.**Options_service_location**
Bases: object

Mixing for providing the –service-location option.

opt_service_location(*value*)
Service location, in the form BASEDN:HOST[:PORT]

postOptions_service_location()

exception ldaptor.usage.**UsageError**
Bases: Exception

Module contents

A Pure-Python Twisted library for LDAP

1.3 Ldaptor Cookbook

1.3.1 Ldaptor Cookbook

The following recipies demonstrate how to accomplish various LDAP-related tasks with Twisted and Ldaptor. Recipies are broken into categories for convenience.

LDAP Clients

The following recipies demonstrate asynchronous LDAP clients.

A Minimal Client Using Endpoints

While Ldaptor exposes helper classes to connect clients to the DIT, it is possible to use the Twisted *endpoints* API to connect an Ldaptor client to a server.

Code

```
#! /usr/bin/env python

import sys

from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols.ldap.ldapsyntax import LDAPEntry
from twisted.internet.defer import inlineCallbacks
from twisted.internet.endpoints import clientFromString, connectProtocol
from twisted.internet.task import react
from twisted.python import log


@inlineCallbacks
def onConnect(clientProtocol):
    o = LDAPEntry(clientProtocol, "dc=fr")
    results = yield o.search()
    data = u"".join([result.getLDIF() for result in results])
    log.msg(u"LDIF formatted results:\n{}".format(data))


def onError(err, reactor):
    if reactor.running:
        log.err(err)
        reactor.stop()

```

(continues on next page)

(continued from previous page)

```
26
27 def main(reactor):
28     log.startLogging(sys.stdout)
29     endpoint_str = "tcp:host=localhost:port=8080"
30     e = clientFromString(reactor, endpoint_str)
31     d = connectProtocol(e, LDAPClient())
32     d.addCallback(onConnect)
33     d.addErrback(onError, reactor)
34     return d
35
36
37 react(main)
```

Discussion

The `twisted.internet.task.react()` function is perfect for running a one-shot *main()* function. When *main()* is called, we create a client endpoint from a string description and the reactor. `twisted.internet.endpoints.connectProtocol()` is used to make a one-time connection to an LDAP directory listening on the local host, port 8080. When the deferred returned from that function fires, the connection has been established and the client protocol instance is passed to the `onConnect()` callback.

This callback uses inline deferreds to make the syntax more compact. We create an *`ldaptor.protocols.ldap.ldapsyntax.LDAPEntry`* with a DN matching the root of the directory and call the asynchronous `search()` method. The result returned when the deferred fires is a list of `LDAPEntry` objects.

When cast as strings, these entries are formatted as LDIF.

Searching with the Paged Search Result Control

Some *DITs* place limits on the number of entries they are willing to return as the result of a LDAP SEARCH request. Microsoft's Active Directory is one such service. In order to query and process large result sets, you can use the paged result control (OID 1.2.840.113556.1.4.319) if you DIT supports it.

The paged result control allows you to request a particular page size. The *DIT* will return a response control that has a magic cookie if the there are additional pages of results. You can use the cookie on a new request to process the results one page at a time.

Code

For *ad.example.com* domain, store the admin password in a file named *pass_file* and run the following example, where *10.20.1.2* is replaced with the IP of your AD server:

```
python docs/source/cookbook/client_paged_search_results.py \
    tcp:host=10.20.1.2:port=389 \
    'CN=Administrator,CN=Users,DC=ad,DC=example,DC=com' \
    pass_file \
    'CN=Users,DC=ad,DC=example,DC=com' \
    --page-size 5
```

The output should look like:

```
Page 1
CN=Users,DC=ad,DC=example,DC=com
CN=Administrator,CN=Users,DC=ad,DC=example,DC=com
CN=Guest,CN=Users,DC=ad,DC=example,DC=com
CN=SUPPORT_388945a0,CN=Users,DC=ad,DC=example,DC=com
CN=HelpServicesGroup,CN=Users,DC=ad,DC=example,DC=com
Page 2
CN=TelnetClients,CN=Users,DC=ad,DC=example,DC=com
CN=krbtgt,CN=Users,DC=ad,DC=example,DC=com
CN=Domain Computers,CN=Users,DC=ad,DC=example,DC=com
There were 8 results returned in total.
```

```
#! /usr/bin/env python

import argparse
import sys

from twisted.internet import defer
from twisted.internet.endpoints import clientFromString, connectProtocol
from twisted.internet.task import react
from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols.ldap.ldapsyntax import LDAPEntry
from ldaptor.protocols import pureber


@defer.inlineCallbacks
def onConnect(client, args):
    binddn = args.bind_dn
    bindpw = args.passwd_file.read().strip()
    if args.start_tls:
        yield client.startTLS()
    try:
        yield client.bind(binddn, bindpw)
    except Exception as ex:
        print(ex)
        raise
    page_size = args.page_size
    cookie = ''
    page = 1
    count = 0
    while True:
        results, cookie = yield process_entry(
            client,
            args,
            args.filter,
            page_size=page_size,
            cookie=cookie)
        count += len(results)
        print("Page {}".format(page))
        display_results(results)
        if len(cookie) == 0:
            break
        page += 1
    print("There were {} results returned in total.".format(count))


@defer.inlineCallbacks
```

(continues on next page)

(continued from previous page)

```
def process_entry(client, args, search_filter, page_size=100, cookie=''):
    basedn = args.base_dn
    control_value = pureber.BERSequence([
        pureber.BERInteger(page_size),
        pureber.BEROctetString(cookie),
    ])
    controls = [('1.2.840.113556.1.4.319', None, control_value)]
    o = LDAPEntry(client, basedn)
    results, resp_controls  = yield o.search(
        filterText=search_filter,
        attributes=['dn'],
        controls=controls,
        return_controls=True)
    cookie = get_paged_search_cookie(resp_controls)
    defer.returnValue((results, cookie))


def display_results(results):
    for entry in results:
        print(entry.dn.getText())


def get_paged_search_cookie(controls):
    """
    Input: semi-parsed controls list from LDAP response;
    list of tuples (controlType, criticality, controlValue).
    Parses the controlValue and returns the cookie as a byte string.
    """
    control_value = controls[0][2]
    ber_context = pureber.BERDecoderContext()
    ber_seq, bytes_used = pureber.berDecodeObject(ber_context, control_value)
    raw_cookie = ber_seq[1]
    cookie = raw_cookie.value
    return cookie


def onError(err):
    err.printDetailedTraceback(file=sys.stderr)


def main(reactor, args):
    endpoint_str = args.endpoint
    e = clientFromString(reactor, endpoint_str)
    d = connectProtocol(e, LDAPClient())
    d.addCallback(onConnect, args)
    d.addErrback(onError)
    return d


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="AD LDAP demo.")
    parser.add_argument(
        "endpoint",
        action="store",
        help="The Active Directory service endpoint. See "
             "https://twistedmatrix.com/documents/current/core/howto/endpoints.html
↪#clients")
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument(
        "bind_dn",
        action="store",
        help="The DN to BIND to the service as.")
    parser.add_argument(
        "passwd_file",
        action="store",
        type=argparse.FileType('r'),
        help="A file containing the password used to log into the service.")
    parser.add_argument(
        "base_dn",
        action="store",
        help="The base DN to start from when searching.")
    parser.add_argument(
        "-f",
        "--filter",
        action='store',
        help='LDAP filter')
    parser.add_argument(
        "-p",
        "--page-size",
        type=int,
        action='store',
        default=100,
        help='Page size (default 100).')
    parser.add_argument(
        "--start-tls",
        action="store_true",
        help="Request StartTLS after connecting to the service.")
    args = parser.parse_args()
    react(main, [args])
```

Discussion

On connecting to the LDAP service, our client establishes TLS and BINDs as a DN that has permission to perform a search. Page, cookie, and the result count are intialized before looping to process each page. Initially, a blank cookie is used in the search request. The cookie obtained from each response is used in the next request, until the cookie is blank. This signals the end of the loop.

Note how the search returns a tuple of results *and* controls from the LDAP response. This is because the *return_controls* flag of the search was set to *True*.

Parsing the cookie requires some *BER* decoding. For details on encoding of the control value, refer to RFC 2696.

Adding an LDAP Entry

Ldaptor allows your LDAP client make many different kinds of LDAP requests. In this example, a simple client connects to an LDAP service and requests adding an new entry.

Code

```
#! /usr/bin/env python

import sys

from twisted.internet import defer
from twisted.internet.endpoints import clientFromString, connectProtocol
from twisted.internet.task import react
from twisted.python import log
from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols import pureber, pureldap


def entry_to_attributes(entry):
    """
    Convert a simple mapping to the data structures required for an
    entry in the DIT.

    Returns: (dn, attributes)
    """
    attributes = {}
    dn = None
    for prop, value in entry.items():
        if prop == 'dn':
            dn = value
            continue
        attributes.setdefault(prop, set()).add(value)
    if dn is None:
        raise Exception("Entry needs to include key, `dn`!")
    ldap_attributes = []
    for attrib, values in attributes.items():
        ldap_attribute_type = pureldap.LDAPAttributeDescription(attrib)
        ldap_attribute_values = []
        for value in values:
            ldap_attribute_values.append(pureldap.LDAPAttributeValue(value))
        ldap_values = pureber.BERSet(ldap_attribute_values)
        ldap_attributes.append((ldap_attribute_type, ldap_values))
    return dn, ldap_attributes


@defer.inlineCallbacks
def onConnect(client, entry):
    dn, attributes = entry_to_attributes(entry)
    op = pureldap.LDAPAddRequest(entry=dn, attributes=attributes)
    response = yield client.send(op)
    if response.resultCode != 0:
        log.err("DIT reported error code {}: {}".format(
            response.resultCode, response.errorMessage))


def onError(err, reactor):
    if reactor.running:
        log.err(err)
        reactor.stop()


```

(continues on next page)

(continued from previous page)

```
def main(reactor):
    log.startLogging(sys.stdout)
    entry = {
        "dn": "gn=Jane+sn=Doe,ou=people,dc=example,dc=fr",
        "c": "US",
        "gn": "Jane",
        "l": "Philadelphia",
        "objectClass": "addressbookPerson",
        "postalAddress": "230",
        "postalCode": "314159",
        "sn": "Doe",
        "st": "PA",
        "street": "Mobius Strip",
        "userPassword": "terces",
    }
    endpoint_str = "tcp:host=localhost:port=8080"
    e = clientFromString(reactor, endpoint_str)
    d = connectProtocol(e, LDAPClient())
    d.addCallback(onConnect, entry)
    d.addErrback(onError, reactor)
    return d


react(main)
```

Discussion

Once again, the `twisted.internet.task.react()` function is used to call the *main()* function of the client. When *main()* is called, we create a client endpoint from a string description and the reactor. `twisted.internet.endpoints.connectProtocol()` is used to make a one-time connection to a LDAP directory listening on the local host, port 8080.

When the deferred returned from that function fires, the connection has been established and the client protocol instance is passed to the `onConnect()` callback, along with our entry.

In this case we use a simple Python dictionary to model our entry. We need to transform this into a data structure that *`ldaptor.protocols.pureldap.LDAPAddRequest`* can use. Once we've created the request, it is relatively simple to send it to the directory service with a call to the *send()* method of our client. The response will indicate either success or failure.

LDAP Proxies

An LDAP proxy sits between an LDAP client and an LDAP server. It accepts LDAP requests from the client and forwards them to the LDAP server. Responses from the server are then relayed back to the client.

Why is it Useful?

An LDAP proxy has many different uses:

- If a client does not natively support LDAP over SSL or StartTLS, a proxy can be run on the client host. The client can interact with the proxy which can use LDAPS or StartTLS when communicating with the backend service.
- When troubleshooting LDAP connections between LDAP clients and servers, it can be useful to determine what kinds of requests and responses are passing between the client and server. Sometimes, access to client or server logs is not available or not helpful. By logging the interactions at the proxy, one can gain insight into what requests are being made by the client and what responses the server makes.
- It may be desirable to provide limited access to an LDAP service. For example, it may be desirable to grant an application search access to an LDAP DIT, but any Modify, Add, or Delete operations are not allowed. A proxy can be configured to disable those particular LDAP operations.
- LDAP requests can be modified before sending them on to the LDAP server. For example, the base DN of search could be transparently modified based on the current BIND user.
- Similarly, LDAP responses from the server can be modified before sending them to the client. For example, search results could be populated with computed attributes, or a domain could be appended to any returned *uid* attribute.
- The proxy can be configured to connect to one of several LDAP servers (replicas). This can be an effective technique when a particular LDAP client library shows affinity for a particular host in an LDAP replica round-robin architecture. The client can be configured to always connect to the proxy, which in turn will distrbute the connections amongst the replicas.

Proxy Recipies

Logging LDAP Proxy

A logging LDAP proxy inspects the LDAP requests and responses and records them in a log.

Code

```
#! /usr/bin/env python

from ldaptor.protocols import pureldap
from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols.ldap.ldapconnector import connectToLDAPEndpoint
from ldaptor.protocols.ldap.proxybase import ProxyBase
from twisted.internet import defer, protocol, reactor
from twisted.python import log
from functools import partial
import sys
```

(continues on next page)

(continued from previous page)

```
class LoggingProxy(ProxyBase):
    """
    A simple example of using `ProxyBase` to log requests and responses.
    """
    def handleProxiedResponse(self, response, request, controls):
        """
        Log the representation of the responses received.
        """
        log.msg("Request => " + repr(request))
        log.msg("Response => " + repr(response))
        return defer.succeed(response)

def ldapBindRequestRepr(self):
    l=[]
    l.append('version={0}'.format(self.version))
    l.append('dn={0}'.format(repr(self.dn)))
    l.append('auth=****')
    if self.tag!=self.__class__.tag:
        l.append('tag={0}'.format(self.tag))
    l.append('sasl={0}'.format(repr(self.sasl)))
    return self.__class__.__name__+'('+', '.join(l)+')'

pureldap.LDAPBindRequest.__repr__ = ldapBindRequestRepr

if __name__ == '__main__':
    """
    Demonstration LDAP proxy; listens on localhost:10389 and
    passes all requests to localhost:8080.
    """
    log.startLogging(sys.stderr)
    factory = protocol.ServerFactory()
    proxiedEndpointStr = 'tcp:host=localhost:port=8080'
    use_tls = False
    clientConnector = partial(
        connectToLDAPEndpoint,
        reactor,
        proxiedEndpointStr,
        LDAPClient)

    def buildProtocol():
        proto = LoggingProxy()
        proto.clientConnector = clientConnector
        proto.use_tls = use_tls
        return proto

    factory.protocol = buildProtocol
    reactor.listenTCP(10389, factory)
    reactor.run()
```

Discussion

The main idea in the above program is to subclass `ldaptor.protocols.ldap.proxybase.ProxyBase` and override its `handleProxiedResponse()` method.

The function `ldapBindRequestRepr()` is used to patch the `__repr__()` magic method of the *ldaptor.protocols.pureldap.LDAPBindRequest* class. The representation normally prints the BIND password, which is typically *not* what you want.

The main program entry point starts logging and creates a generic server factory. The proxied LDAP server is configured to run on the local host on port 8080. The factory protocol is set to a function that takes no arguments and returns an instance of our `LoggingProxy` that has been configured with a *clientConnector* callable. When this callable is invoked, it will return a deferred that will fire with a `LDAPClient` instance when a connection to the proxied LDAP server is established. The Twisted reactor is then configured to listen on TCP port 10389 and use the factory to create server protocol instances to handle incoming connections.

The `ProxyBase` class handles the typical LDAP protocol events but provides convenient hooks for intercepting LDAP requests and responses. In this proxy, we wait until we have a reponse and log both the request and the response. in the case of a search request with multiple responses, the request is repeatedly displayed with each response.

This program explicitly starts logging and the Twisted reactor loop. However, the **`twistd`** program can perform these tasks for you and allow you to configure options from the command line.

```
from ldaptor.protocols import pureldap
from ldaptor.protocols.ldap.ldapclient import LDAPClient
from ldaptor.protocols.ldap.ldapconnector import connectToLDAPEndpoint
from ldaptor.protocols.ldap.proxybase import ProxyBase
from twisted.application.service import Application, Service
from twisted.internet import defer, protocol, reactor
from twisted.internet.endpoints import serverFromString
from twisted.python import log
from functools import partial

class LoggingProxy(ProxyBase):
    """
    A simple example of using `ProxyBase` to log requests and responses.
    """
    def handleProxiedResponse(self, response, request, controls):
        """
        Log the representation of the responses received.
        """
        log.msg("Request => " + repr(request))
        log.msg("Response => " + repr(response))
        return defer.succeed(response)


def ldapBindRequestRepr(self):
    l=[]
    l.append('version={0}'.format(self.version))
    l.append('dn={0}'.format(repr(self.dn)))
    l.append('auth=****')
    if self.tag!=self.__class__.tag:
        l.append('tag={0}'.format(self.tag))
    l.append('sasl={0}'.format(repr(self.sasl)))
    return self.__class__.__name__+'('+', '.join(l)+')'

pureldap.LDAPBindRequest.__repr__ = ldapBindRequestRepr
```

(continues on next page)

(continued from previous page)

```
class LoggingProxyService(Service):
    endpointStr = "tcp:10389"
    proxiedEndpointStr = 'tcp:host=localhost:port=8080'

    def startService(self):
        factory = protocol.ServerFactory()
        use_tls = False
        proxiedEndpointStr = 'tcp:host=localhost:port=8080'
        clientConnector = partial(
            connectToLDAPEndpoint,
            reactor,
            self.proxiedEndpointStr,
            LDAPClient)

        def buildProtocol():
            proto = LoggingProxy()
            proto.clientConnector = clientConnector
            proto.use_tls = use_tls
            return proto

        factory.protocol = buildProtocol
        ep = serverFromString(reactor, self.endpointStr)
        d = ep.listen(factory)
        d.addCallback(self.setListeningPort)
        d.addErrback(log.err)

    def setListeningPort(self, port):
        self.port_ = port

    def stopService(self):
        # If there are asynchronous cleanup tasks that need to
        # be performed, add deferreds for them to `async_tasks`.
        async_tasks = []
        if self.port_ is not None:
            async_tasks.append(self.port_.stopListening())
        if len(async_tasks) > 0:
            return defer.DeferredList(async_tasks, consumeErrors=True)


application = Application("Logging LDAP Proxy")
service = LoggingProxyService()
service.setServiceParent(application)
```

This program is very similar to the previous one. However, this one is run with **twistd**:

```
$ twistd -ny loggingproxy.py
```

The **twistd** program looks for the global name *application* in the script and runs all the services attached to it. We moved most of the startup code from the *if __name__ == '__main__'* block into the service's `startService()` method. This method is called when our service starts up. Conversely, `stopService()` is called when the service is about to shut down.

This improved example also makes use of endpoint strings. These strings are textual descriptions of client and server sockets on which our LDAP proxy server will connect and listen, respectively.

The advantage of endpoints is that you can read these strings from a configuration file and change how your server listens or how you client connects. Our example listens on a plain old TCP socket, but you could easilly switch to a

TLS socket or a UNIX domain socket without having to change a line of code.

Listening on an endpoint is an asynchronous task, so we set a callback to record the listening port. When the service stops, we ask the port to stop listening.

LDAP Merger

A merger forwards search requests to multiple LDAP Servers, and returns the result entries of each successful response.

Usecase

You have multiple LDAP Servers, and you want to combine the search results of all of them. This can be the case if you have an application which needs extra users which are not inside the LDAP directory of your enterprise server, and it is not desired to store them in it. In this case you could use an internal LDAP server on the local filesystem (you can also do this with ldaptor, please look at the LDAP Servers section, File-System LDAP DIT), and combine this server with your general LDAP server.

Caveats

Be aware that it the merger is a read-only implementation: only BIND and SEARCH operations are supported. Beyond that, notice that when binding only the servers where the bind has been successful are delivering search results. So in order to retrieve results on all servers, the bind user must be available on all LDAP servers.

Usage

Code

Store the python code in a file called ldap-merger.tac:

```
#! /usr/bin/env python

from twisted.application import service, internet
from twisted.internet import protocol
from ldaptor.config import LDAPConfig
from ldaptor.protocols.ldap.merger import MergedLDAPServer

application = service.Application("LDAP Merger")

configs = [LDAPConfig(serviceLocationOverrides={"": ('external', 389)}),
           LDAPConfig(serviceLocationOverrides={"": ('localhost', 38942)})]
use_tls = [True, False]
factory = protocol.ServerFactory()
factory.protocol = lambda: MergedLDAPServer(configs, use_tls)
mergeService = internet.TCPServer(389, factory)
mergeService.setServiceParent(application)
```

Discussion

We use two ldap servers: one listening on the host ''external'' on the default port 389, and the other is a server running on localhost with port 38942. TLS is used for the connection to the external server. The merger itself listens on port 389.

Run it with $ twistd -y ldap-merger.tac

LDAP Servers

An LDAP directory information tree (DIT) is a highly specialized database with entries arranged in a tree-like structure.

File-System LDAP DIT

A minimal LDAP DIT that stores entries in the local file system

Code

First, a module that defines our DIT entries– `schema.py`

```
# -*- coding: utf-8 -*-

COUNTRY = (
    'dc=fr',
    {
        'objectClass': ['dcObject','country'],
        'dc': ['fr'],
        'description': ["French country 2 letters iso description"],
    }
)
COMPANY = (
    'dc=example',
    {
        'objectClass': ['dcObject','organization'],
        'dc': ['example'],
        'description': ["My organisation"],
        'o': ["Example, Inc"],
    }
)
PEOPLE = (
    'ou=people',
    {
        'ou': ['people'],
        'description': ['People from Example Inc'],
        'objectclass': ['organizationalunit'],
    }
```

(continues on next page)

(continued from previous page)

```
)
USERS = [
        (
            'uid=yoen',
            {
                'objectClass': ['people', 'inetOrgPerson'],
                'cn': ['Yoen Van der Weld'],
                'sn': ['Van der Weld'],
                'givenName': ['Yoen'],
                'uid': ['yoen'],
                'mail': ['/home/yoen/mailDir'],
                'userPassword': ['secret']
            }
        ),
        (
            'uid=esteban',
            {
                'objectClass': ['people', 'inetOrgPerson'],
                'cn': ['Esteban Garcia Marquez'],
                'sn': ['Garcia Marquez'],
                'givenName': ['Esteban'],
                'uid': ['esteban'],
                'mail': ['/home/esteban/mailDir'],
                'userPassword': ['secret2']
            }
        ),
        (
            'uid=mohamed',
            {
                'objectClass': ['people', 'inetOrgPerson'],
                'cn': ['Mohamed Al Ghâlib'],
                'sn': ['Al Ghâlib'],
                'givenName': ['mohamed'],
                'uid': ['mohamed'],
                'mail': ['/home/mohamed/mailDir'],
                'userPassword': ['secret3']
            }
        ),
    ]
```

Next, the server code– `ldaptor_basic.py`

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

"""
    Testing a simple ldaptor ldap server
    Base on an example by Gaston TJEBBES aka "tonthon":
    http://tonthon.blogspot.com/2011/02/ldaptor-ldap-with-twisted-server-side.html
"""

import tempfile, sys

from twisted.application import service
from twisted.internet import reactor
from twisted.internet.protocol import ServerFactory
from twisted.python.components import registerAdapter
```

(continues on next page)

(continued from previous page)

```
from twisted.python import log
from ldaptor.interfaces import IConnectedLDAPEntry
from ldaptor.protocols.ldap.ldapserver import LDAPServer
from ldaptor.ldiftree import LDIFTreeEntry

from schema import COUNTRY, COMPANY, PEOPLE, USERS


class Tree(object):

    def __init__(self):
        dirname = tempfile.mkdtemp('.ldap', 'test-server', '/tmp')
        self.db = LDIFTreeEntry(dirname)
        self.init_db()

    def init_db(self):
        """
            Add subtrees to the top entry
            top->country->company->people
        """
        country = self.db.addChild(COUNTRY[0], COUNTRY[1])
        company = country.addChild(COMPANY[0], COMPANY[1])
        people = company.addChild(PEOPLE[0], PEOPLE[1])
        for user in USERS:
            people.addChild(user[0], user[1])


class LDAPServerFactory(ServerFactory):
    """
        Our Factory is meant to persistently store the ldap tree
    """
    protocol = LDAPServer

    def __init__(self, root):
        self.root = root

    def buildProtocol(self, addr):
        proto = self.protocol()
        proto.debug = self.debug
        proto.factory = self
        return proto


if __name__ == '__main__':
    if len(sys.argv) == 2:
        port = int(sys.argv[1])
    else:
        port = 8080
    # First of all, to show logging info in stdout :
    log.startLogging(sys.stderr)
    # We initialize our tree
    tree = Tree()
    # When the ldap protocol handle the ldap tree,
    # it retrieves it from the factory adapting
    # the factory to the IConnectedLDAPEntry interface
    # So we need to register an adapter for our factory
    # to match the IConnectedLDAPEntry
```

(continues on next page)

(continued from previous page)

```
    registerAdapter(
        lambda x: x.root,
        LDAPServerFactory,
        IConnectedLDAPEntry)
    # Run it !!
    factory = LDAPServerFactory(tree.db)
    factory.debug = True
    application = service.Application("ldaptor-server")
    myService = service.IServiceCollection(application)
    reactor.listenTCP(port, factory)
    reactor.run()
```

LDAP Server which allows BIND with UPN

The LDAP server implemented by Microsoft Active Directory allows using the UPN as the BIND DN.

It is possible to implement something similar using ldaptor.

Below is a proof-of-concept implementation, which should not be used for production as it has an heuristic method for detecting which BIND DN is an UPN.

handle_LDAPBindRequest is the method called when a BIND request is received.

```
"""
An ldaptor LDAP server which can authenticate based on UPN, as AD does.

The LDAP entry needs to have the `userPrincipalName` attribute set.

dn: uid=bob,ou=people,dc=example,dc=org
objectclass: top
objectclass: person
objectClass: inetOrgPerson
uid: bob
cn: bobby
gn: Bob
sn: Roberts
mail: bob@example.org
homeDirectory: e:\\Users\\bob
userPassword: pass
userPrincipalName: bob@ad.example.org
"""
from __future__ import absolute_import

from ldaptor import interfaces
from ldaptor.protocols import pureldap
from ldaptor.protocols.ldap import distinguishedname, ldaperrors
from twisted.internet import defer
from ldaptor.protocols.ldap.ldapserver import LDAPServer


class LDAPServerWithUPNBind(LDAPServer):
    """
    An LDAP server which support BIND using UPN similar to AD.
    """
    _loginAttribute = b'userPrincipalName'

```

(continues on next page)

(continued from previous page)

```
    def handle_LDAPBindRequest(self, request, controls, reply):
        if request.version != 3:
            raise ldaperrors.LDAPProtocolError(
                'Version %u not supported' % request.version)

        self.checkControls(controls)

        if request.dn == b'':
            # anonymous bind
            self.boundUser = None
            return pureldap.LDAPBindResponse(resultCode=0)

        root = interfaces.IConnectedLDAPEntry(self.factory)

        def _gotUPNResult(results):
            if len(results) != 1:
                # Not exactly one result, so this might not be an UNP.
                return distinguishedname.DistinguishedName(request.dn)

            # A single result, so the UPN might exist.
            return results[0].dn

        if b'@' in request.dn and b',' not in request.dn:
            # This might be an UPN request.
            filterText = b'(' + self._loginAttribute + b'=' + request.dn + b')'
            d = root.search(filterText=filterText)
            d.addCallback(_gotUPNResult)
        else:
            d = defer.succeed(distinguishedname.DistinguishedName(request.dn))

        # Once the BIND DN is known, search for the LDAP entry.
        d.addCallback(lambda dn: root.lookup(dn))

        def _noEntry(fail):
            """
            Called when the requested BIND DN was not found.
            """
            fail.trap(ldaperrors.LDAPNoSuchObject)
            return None
        d.addErrback(_noEntry)

        def _gotEntry(entry, auth):
            """
            Called when the requested BIND DN was found.
            """
            if entry is None:
                raise ldaperrors.LDAPInvalidCredentials()

            d = entry.bind(auth)

            def _cb(entry):
                """
                Called when BIND operation was successful.
                """
                self.boundUser = entry
                msg = pureldap.LDAPBindResponse(
                    resultCode=ldaperrors.Success.resultCode,
```

(continues on next page)

(continued from previous page)

```
                    matchedDN=entry.dn.getText())
                return msg
            d.addCallback(_cb)
            return d
        d.addCallback(_gotEntry, request.auth)

        return d
```

CHAPTER

TWO

META

2.1 Changelog

2.1.1 20.0.0 (2020-09-30)

Changes

- The next release v20.1.0 will drop support for Python 2, and require Python~=3.5
- PyPI release is now done via GitHub Action
- the ldaptor whl is now built with pep517.
- the ldaptor whl is tested with tox. The sdist is now untested, deprecated and should only be used for compatability with very old packaging tools.
- the setup.py file is deprecated and will be removed in a future release.

Bugfixes

- SASL Bind without credentials caused list index out of range. Issue #157.
- ldaptor.protocols.ldap.ldapserver.LDAPServer.handle_LDAPSearchRequest now returns an LDAPSearchResultEntry without any attributes when there is no match between the requested attributes and the entrie's attributes. Issue #166.

2.1.2 Release 19.1 (2019-09-09)

Features

- Basic implementation of `ldaptor.protocols.pureldap.LDAPSearchResultReference`.
- Explicit `ldaptor.protocols.ldap.ldaperrors` classes declaration was made to allow syntax highlighting for this module.
- Example of using LDAP server with the database. Employees are store in the database table and retrieved on server initialization.

Changes

- `ldaptor.protocols.pureldap.LDAPPasswordModifyRequest` string representation now contains `userIdentity`, `oldPasswd` and `newPasswd` attributes. Password attributes are represented as asterisks.
- `ldaptor.protocols.pureldap.LDAPBindRequest` string representation is now using asterisks to represent `auth` attribute.

Bugfixes

- `DeprecationWarning` stacklevel was set to mark the caller of the deprecated methods of the `ldaptor._encoder` classes.
- `NotImplementedError` for `ldaptor.protocols.pureldap.LDAPSearchResultReference` was fixed.
- Regression bug with `LDAPException` instances was fixed (`ldaptor.protocols.ldap.ldapclient` exceptions failed to get their string representations).
- StartTLS regression bug was fixed: `ldaptor.protocols.pureldap.LDAPStartTLSRequest.oid` and `ldaptor.protocols.pureldap.LDAPStartTLSResponse.oid` must be of bytes type.
- `ldaptor.protocols.pureldap` and `ldaptor.protocols.pureber` string representations were fixed: *LDAPResult(resultCode=0, matchedDN='uid=user')* instead of *LDAPResult(resultCode=0, matchedDN="b'uid=user'").*
- `ldaptor.protocols.pureldap.LDAPMatchingRuleAssertion` initialization for Python 3 was failed for bytes arguments.
- `ldaptor.protocols.pureldap.LDAPExtendedResponse` custom tag parameter was not used.
- `ldaptor._encoder.to_bytes()` was fixed under Python 3 to return integers as their numeric representation rather than a sequence of null bytes.

2.1.3 Release 19.0 (2019-03-05)

Features

- Ability to logically compare ldaptor.protocols.pureldap.LDAPFilter_and and ldaptor.protocols.pureldap.LDAPFilter_or objects with ==.
- Ability to customize ldaptor.protocols.pureldap.LDAPFilter_* object's encoding of values when using asText.
- New client recipe- adding an entry to the DIT.
- Ability to use paged search control for LDAP clients.
- New client recipie- using the paged search control.

Changes

- Using modern classmethod decorator instead of old-style method call.
- Usage of zope.interfaces was updated in preparation for python3 port.
- `toWire` method is used to get bytes representation of *ldaptor* classes instead of `__str__` which is deprecated now.
- Code was updated to pass *python3 -m compileall* in preparation for py3 port.
- Code is linted under python 3 in preparation for py3 port.
- Continuous test are executed only against latest related Twisted and latest Twisted trunk branch.
- The local development environment was updated to produce overall and diff coverage reports in HTML format.
- *six* package is now a direct dependency in preparation for the Python 3 port, and has replaced the ldaptor.compat module.
- Remove Python 3.3 from tox as it is EOL.
- Add API documentation for `LDAPAttributeSet` and `startTLS`.
- Quick start and cookbook examples were moved to separate files and made agnostic to the Python version.
- dependency on pyCrypto replaced with pure python passlib.
- replace direct dependency on pyOpenSSL with Twisted[tls]

Bugfixes

- DN matching is now case insensitive.
- Proxies now terminate the connection to the proxied server in case a client immediately closes the connection.
- asText() implemented for LDAPFilter_extensibleMatch
- Children of `ldaptor.inmemory.ReadOnlyInMemoryLDAPEntry` subclass instances are added as the same class instances.
- Redundant attributes keys sorting was removed from `ldaptor.entry.BaseLDAPEntry` methods.

2.1.4 Release 16.0 (2016-06-07)

Features

- Make meta data introspectable
- Added *proxybase.py*, an LDAP proxy that is easier to hook into.
- When parsing LDAPControls, criticality may not exist while controlValue still does
- Requested attributes can also be passed as '*' symbol
- Numerous small bug fixes.
- Additional documentation
- Updated Travis-CI, Tox and other bits for better coverage.

2.1.5 Release 14.0 (2014-10-31)

Ldaptor has a new version schema. As a first-party library we now follow Twisted's example.

License

- Ldaptor's original author Tommi Virtanen changed the license to the MIT (Expat) license.
- ldaptor.md4 has been replaced by a 3-clause BSD version.

API Changes

- Ldaptor client and server: None
- Everything having to do with webui and Nevow have been *removed*.

Features

- Travis CI is now used for continuous integration.
- Test coverage is now measured. We're currently at around 75%.
- tox is used now to test ldaptor on all combinations of pypy, Python 2.6, Python 2.7 and Twisted versions from 10.0 until 14.0.
- A few ordering bugs that were exposed by that and are fixed now.
- ldaptor.protocols.pureldap.LDAPExtendedRequest now has additional tests.
- The new ldaptor.protocols.pureldap.LDAPAbandonRequest adds support for abandoning requests.
- ldaptor.protocols.pureldap.LDAPBindRequest has basic SASL support now. Higher-level APIs like ldapclient don't expose it yet though.

Bugfixes

- ldaptor.protocols.ldap.ldapclient's now uses log.msg for it's debug listing instead of the non-Twisted log.debug.
- String literal exceptions have been replaced by real Exceptions.
- "bin/ldaptor-ldap2passwd –help" now does not throws an exception anymore (debian bug #526522).
- ldaptor.delta.Modification and ldaptor.protocols.ldap.ldapsyntax.PasswordSetAggregateError that are used for adding contacts now handle unicode arguments properly.
- ldaptor.protocols.pureldap.LDAPExtendedRequest's constructor now handles STARTTLS in accordance to RFC2251 so the constructor of ldaptor.protocols.pureldap.LDAPStartTLSRequest doesn't fail anymore.
- ldaptor.protocols.ldap.ldapserver.BaseLDAPServer now uses the correct exception module in dataReceived.
- ldaptor.protocols.ldap.ldaperrors.LDAPException: "Fix deprecated exception error"
- bin/ldaptor-find-server now imports dns from the correct twisted modules.
- bin/ldaptor-find-server now only prints SRV records.
- ldaptor.protocols.ldap.ldapsyntax.LDAPEntryWithClient now correctly propagates errors on search(). The test suite has been adapted appropriately.

- ldaptor.protocols.ldap.ldapconnector.LDAPConnector now supports specifying a local address when connecting to a server.
- The new ldaptor.protocols.pureldap.LDAPSearchResultReference now prevents ldaptor from choking on results containing SearchResultReference (usually from Active Directory servers). It is currently only a stub and silently ignored.
- hashlib and built-in set() are now used instead of deprecated modules.

Improved Documentation

- Added, updated and reworked documentation using Sphinx. Dia is required for converting diagrams to svg/png, this might change in the future.
- Dia is now invoked correctly for diagram generation in a headless environment.
- The documentation is now hosted on https://ldaptor.readthedocs.org/.

2.1.6 Prehistory

All versions up to and including 0.0.43 didn't have a changelog.

2.2 Status and History

Ldaptor was created by Tommi Virtanen who developed it during the years 2001-2008. From 2007 and onwards mainly bug fixes were added, many contributed by Debian maintainers. Development picked back up in 2014 by Bret Curtis with Tommi's consent and was migrated to Twisted where it is a first-party Twisted library. Ldaptor can be found here:

https://github.com/twisted/ldaptor

The LDAP client library functionality is in active use. It is stable and works very well.

2.3 Contributions

2.3.1 How to Contribute

Head over to: https://github.com/twisted/ldaptor and submit your bugs or feature requests.

If you wish to contribute code, just fork it, make a branch and send us a pull request. We'll review it, and push back if necessary.

Check docs/PULL_REQUEST_TEMPLATE.md for more info about how to pull request process.

Ldaptor generally follows the coding and documentation standards of the Twisted project.

Development environment

Tox is used to manage both local development and CI environment.

The recommended local dev enviroment is *tox -e py27-test-dev*

When running on local dev env, you will get a coverage report for whole code as well as for the changes since *master*. The reports are also produced in HTML at:

- build/coverage-html/index.html
- build/coverage-diff.html

You can run a subset of the test by passing the dotted path to the test or test case, test module or test package:

```
tox -e py27-test-dev ldaptor.test.test_delta.TestModifyOp.testAsLDIF
tox -e py27-test-dev ldaptor.test.test_usage
```

Release notes

To simplify the release process each change should be recorded into the docs/source/NEWS.rst in a wording targeted to end users. Try not to write the release notes as a commit message.

Release process

The release is done automatically via GitHub actions when a new tag is pushed. A new tag can be pushed with:

```
pipx run --spec="zest.releaser[recommended]>=6.22.1" fullrelease
```

PyPI access is done via the HTTP API token stored in GitHub Secrets as PYPI_GITHUB_PACKAGE_UPLOAD from https://github.com/twisted/ldaptor/settings/secrets

You can test the release process (without the publish) using *tox -e release*. Inspect the distributable files with *tree dist*, you could upload them with *twine*.

Building the documentation

The documentation is managed using Python Sphinx and is generated in docs/build.

There is a helper to build the documentation using tox

```
tox -e documentation
```

2.3.2 Contributors

- Anton Gyllenberg
- Aren Sandersen
- Bret Curtis
- Carl Waldbieser
- Christopher Bartz
- David Strauss

- HawkOwl
- Hynek Schlawack
- Kenny MacDermid
- Michael Schlenker
- Sergey Shubin
- Stefan Andersson
- Tommi Virtanen

2.4 Glossary

BER *Basic Encoding Rules*. A set of Abstract Syntax Notation One (ASN.1) encoding rules that are used to create the binary representation of LDAP protocol messages "on the wire".

DIT

DITs *Directory Information Tree*. A tree-like representation of entries an LDAP service presents to clients.

LDIF The LDAP Data Interchange Format. A plain text data format that can represent directory content or an update request.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W