eldap

eldap 1.0.1
July 10, 2013

July 10, 2013

1 Eldap User's Guide

The *Eldap* application provides an api for accessing an LDAP server.

The original code was developed by Torbjörn Törnkvist.

2 Reference Manual

The *Eldap* application provides an api for accessing an LDAP server.

The original code was developed by Torbjörn Törnkvist.

eldap

Erlang module

This module provides a client api to the Lightweight Directory Access Protocol (LDAP).

References:

- RFC 4510 - RFC 4519

The above publications can be found at **IETF**.

Types

```
handle()    Connection handle
attribute() {Type = string(), Values=[string()]}
modify_op() See mod_add/2, mod_delete/2, mod_replace/2
scope()     See baseObject/0, singleLevel/0, wholeSubtree/0
dereference() See neverDerefAliases/0, derefInSearching/0, derefFindingBaseObj/0, derefAlways/0
filter()    See present/1, substrings/2,
                equalityMatch/2, greaterOrEqual/2, lessOrEqual/2,
                approxMatch/2,
                'and'/1, 'or'/1, 'not'/1.
```

Exports

`open([Host]) -> {ok, Handle} | {error, Reason}`

Types:

`Handle = handle()`

Setup a connection to an LDAP server, the `HOST`'s are tried in order.

`open([Host], [Option]) -> {ok, Handle} | {error, Reason}`

Types:

`Handle = handle()`

`Option = {port, integer()} | {log, function()} | {timeout, integer()} | {ssl, boolean()} | {sslopts, list()}`

Setup a connection to an LDAP server, the `HOST`'s are tried in order.

The log function takes three arguments, `fun(Level, FormatString, [FormatArg]) end`.

Timeout set the maximum time in milliseconds that each server request may take.

`close(Handle) -> ok`

Types:

`Handle = handle()`

Shutdown the connection.

`simple_bind(Handle, Dn, Password) -> ok | {error, Reason}`

Types:

Handle = handle()
Dn = string()
Password = string()

Authenticate the connection using simple authentication.

`add(Handle, Dn, [Attribute]) -> ok | {error, Reason}`

Types:

Handle = handle()
Dn = string()
Attribute = attribute()

Add an entry. The entry must not exist.

```
add(Handle,
    "cn=Bill Valentine, ou=people, o=Example Org, dc=example, dc=com",
     [{"objectclass", ["person"]},
      {"cn", ["Bill Valentine"]},
      {"sn", ["Valentine"]},
      {"telephoneNumber", ["545 555 00"]}]
   )
```

`delete(Handle, Dn) -> ok | {error, Reason}`

Types:

Dn = string()

Delete an entry.

```
delete(Handle, "cn=Bill Valentine, ou=people, o=Example Org, dc=example, dc=com")
```

`mod_add(Type, [Value]) -> modify_op()`

Types:

Type = string()
Value = string()

Create an add modification operation.

`mod_delete(Type, [Value]) -> modify_op()`

Types:

Type = string()
Value = string()

Create a delete modification operation.

`mod_replace(Type, [Value]) -> modify_op()`

Types:

Type = string()

Value = string()

Create a replace modification operation.

`modify(Handle, Dn, [ModifyOp]) -> ok | {error, Reason}`

Types:

Dn = string()

ModifyOp = modify_op()

Modify an entry.

```
modify(Handle, "cn=Bill Valentine, ou=people, o=Example Org, dc=example, dc=com",
       [eldap:mod_replace("telephoneNumber", ["555 555 00"]),
 eldap:mod_add("description", ["LDAP Hacker"]) ])
```

`modify_dn(Handle, Dn, NewRDN, DeleteOldRDN, NewSupDN) -> ok | {error, Reason}`

Types:

Dn = string()

NewRDN = string()

DeleteOldRDN = boolean()

NewSupDN = string()

Modify the DN of an entry. `DeleteOldRDN` indicates whether the current RDN should be removed after operation. `NewSupDN` should be "" if the RDN should not be moved or the new parent which the RDN will be moved to.

```
modify_dn(Handle, "cn=Bill Valentine, ou=people, o=Example Org, dc=example, dc=com ",
          "cn=Bill Jr Valentine", true, "")
```

`search(Handle, SearchOptions) -> {ok, #eldap_search_result{}} | {error, Reason}`

Types:

SearchOptions = #eldap_search{} | [SearchOption]

SearchOption = {base, string()} | {filter, filter()} | {scope, scope()} | {attributes, [string()]} | {deref, dereference()} | | {types_only, boolean()} | {timeout, integer()}

Search the directory with the supplied the SearchOptions. The base and filter options must be supplied. Default values: scope is `wholeSubtree()`, deref is `derefAlways()`, types_only is `false` and timeout is `0` (meaning infinity).

```
Filter = eldap:substrings("cn", [{any,"V"}]),
search(Handle, [{base, "dc=example, dc=com"}, {filter, Filter}, {attributes, ["cn"]}]),
```

`baseObject() -> scope()`

Search baseobject only.

`singleLevel() -> scope()`

Search the specified level only, i.e. do not recurse.

`wholeSubtree() -> scope()`

Search the entire subtree.

`neverDerefAliases() -> dereference()`

Never derefrence aliases, treat aliases as entries.

`derefAlways() -> dereference()`

Always derefrence aliases.

`derefInSearching() -> dereference()`

Derefrence aliases only when searching.

`derefFindingBaseObj() -> dereference()`

Derefrence aliases only in finding the base.

`present(Type) -> filter()`

Types:

`Type = string()`

Create a filter which filters on attribute type presence.

`substrings(Type, [SubString]) -> filter()`

Types:

`Type = string()`
`SubString = {StringPart, string()}`
`StringPart = initial | any | final`

Create a filter which filters on substrings.

`equalityMatch(Type, Value) -> filter()`

Types:

`Type = string()`
`Value = string()`

Create a equality filter.

`greaterOrEqual(Type, Value) -> filter()`

Types:

`Type = string()`
`Value = string()`

Create a greater or equal filter.

`lessOrEqual(Type, Value) -> filter()`

Types:

`Type = string()`
`Value = string()`

Create a less or equal filter.

`approxMatch(Type, Value) -> filter()`

Types:

`Type = string()`
`Value = string()`

Create a approximation match filter.

`'and'([Filter]) -> filter()`

Types:

`Filter = filter()`

Creates a filter where all `Filter` must be true.

`'or'([Filter]) -> filter()`

Types:

`Filter = filter()`

Create a filter where at least one of the `Filter` must be true.

`'not'(Filter) -> filter()`

Types:

`Filter = filter()`

Negate a filter.