

Programming with Net::LDAP

Graham Barr
<gbarr@pobox.com>

What is a directory ?

- Specialized database
 - Structured
 - Distributed
 - Typed information
 - Text strings
 - JPEG pictures
 - X.509 certificates

X.500

- X.500 was the first standard directory
- Developed by the ITU/ISO groups
 - work began as early as 1979
- Well thought out design containing many good ideas
- Very complex
- Required powerful computers for its time
- Defines things like inter-server communications, access controls

Slide 3

LDAP

- Developed to overcome the complexities and heavyweightness of X.500 DAP
- LDAP is a protocol, many early implementations were just gateways to X.500 directories
- Designed to provide 90% of the X.500 functionality
- Most X.500 products now come with an LDAP gateway
- LDAP working groups are working to reproduce all other X.500 functionality via extensions

Slide 4

Structure

- The Directory Information Tree (DIT) is made up of objects called entries
- Each entry is composed of attributes which contain the information recorded about each object
- Entries are organized into a tree structure
- Each entry is uniquely identified by a Distinguished Name (DN)
- The DN of an entry is defined by the entry's position within the tree structure

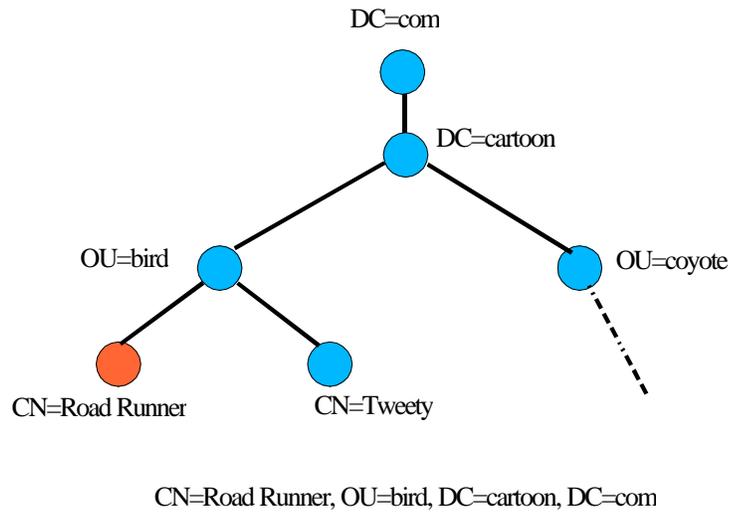
Slide 5

Distinguished Names

- A DN is made up of one or more Relative Distinguished Names (RDN)
- Each RDN is separated by a comma
 - LDAP version 2 allowed a semi-colon
- Example
 - `CN=Road Runner, OU=bird, DC=cartoon, DC=com`
 - The RDN's are
 - `CN=Road Runner`
 - `OU=bird`
 - `DC=cartoon`
 - `DC=com`

Slide 6

Structure Example



Slide 7

Entry Attributes

- The attributes an entry must or may have is defined by either
 - Content rules on a per-server basis
 - objectClass attribute and a schema on the server
- The objectClass attribute is a multi-valued attribute
- Each objectClass defines a list of attributes that the entry must or may have

Slide 8

Attributes

- Attributes are defined in the server schema
- Properties that can be defined are
 - Single or Multi-valued
 - Types of searches that can be performed on them
 - Type of data stored in them
 - Minimum length of available storage
 - Alias names
 - Description

Slide 9

Attribute schema example

```
( 2.5.4.41
  NAME 'name'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15{32768} )

( 2.5.4.4
  NAME ( 'sn' $ 'surname' )
  SUP name )
```

Slide 10

objectClass schema example

```
( 2.5.6.6 NAME 'person' SUP top STRUCTURAL
  MUST ( sn $ cn )
  MAY ( userPassword $ telephoneNumber $ seeAlso $
        description ) )

( 2.16.840.1.113730.3.2.2
  NAME 'inetOrgPerson'
  DESC 'RFC2798: Internet Organizational Person'
  SUP organizationalPerson
  STRUCTURAL
  MAY ( audio $ businessCategory $ carLicense $
        departmentNumber $ displayName $
        employeeNumber $ employeeType $ givenName $
        homePhone $ homePostalAddress $ initials $
        jpegPhoto $ labeledURI $ mail $ manager $
        mobile $ o $ pager $ photo $ roomNumber $
        secretary $ uid $ userCertificate $
        x500uniqueIdentifier $ preferredLanguage $
        userSMIMECertificate $ userPKCS12 ) )
```

Slide 11

Access Control

- There are various levels of access control
 - The directory manager has access to everything
 - Users may be able to modify their own entry
 - Attributes may have permissions on them

Slide 12

Directory Uses

- A directory can be used in many ways
 - Employee database
 - Equipment inventory
 - List of groups
 - Email list
 - Organizational groups
 - NIS replacement
 - Software distribution

Slide 13

Connecting

- Connection is performed as part of the object constructor

```
$ldap = Net::LDAP->new($hostname);
```

- Options can be passed after the hostname

```
$ldap = Net::LDAP->new($hostname,  
    port    => $port, # default: 389  
    debug   => 3,     # default: 0  
    timeout => 60,    # default: 120  
    version => 3      # default: 2  
);
```

Slide 14

Authentication

- Referred to a bind-ing
- LDAP supports several methods
 - Anonymous
 - Simple
 - SASL

Slide 15

Anonymous Authentication

- Supported in both version 2 and 3
- ```
$r = $ldap->bind;
```
- Not required in version 3, but...
- ```
$r = $ldap->bind(version => 3);
```

Slide 16

Simple Authentication

- Supported in both version 2 and 3

```
$r = $ldap->bind($DN, password => $pw);
```

- This method of authentication passes your password in **CLEAR** text over the network

Slide 17

SASL Authentication

- SASL is a framework for providing challenge-response authentication
- Requires version 3 server

```
use Authn::SASL;  
$sas1 = Authn::SASL->new( 'CRAM-MD5'  
    password => $password  
    );  
$r = $ldap->bind($DN,  
    sasl => $sas1,  
    version => 3  
    );
```

Slide 18

Return Values

- Most methods in Net::LDAP return an object, this object provides methods to obtain the results of the operation that was performed
- A result code is returned by the method `->code`
- An error message is returned by the method `->error`
- In most cases a success gives a result code of zero

```
warn $r->error if $r->code != LDAP_SUCCESS;
```

Slide 19

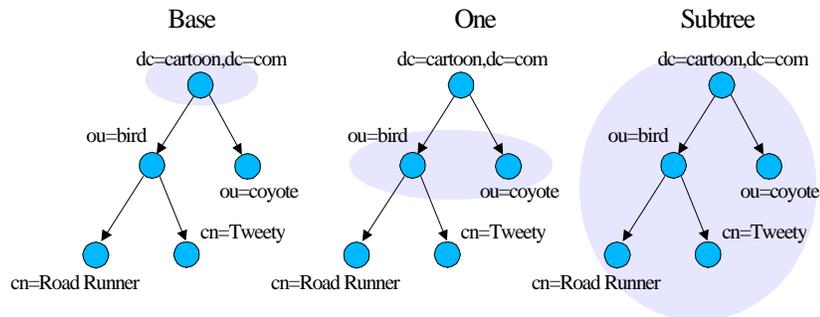
Searching

- To perform a search there are three basic things you must know.
 - Search base
 - The name of a DN within the directory where the search will begin
 - Scope
 - How to perform the search
 - Filter
 - The criteria an entry must match to be returned

Slide 20

Scope

- LDAP servers support three different scopes for searching



Slide 21

Filter

- A filter is the criteria that an entry must match to be returned
- Boolean expression of assertions on the attributes of the entry
 - Examples

```
(&(ou=bird)(cn=*Runner))  
(objectClass=Person)  
(&(objectClass=ipService)  
(cn=ldap)(ipServiceProtocol=tcp))
```

Slide 22

Basic Filter Syntax

- LDAP filters use a prefix syntax, described in RFC2254
 - Equality, existence and sub-string

```
(cn=Road Runner)
(cn=*)
(cn=Road*)
(cn=*Runner)
(cn=R*R*r)
```
 - Relational

```
(age>=18)
(age<=16)
```
 - Approximate

```
(weight~=180)
```

Slide 23

Complex Filters

- Filters may be combined using simple boolean logic
 - And

```
(&(ou=bird)(cn=Road Runner)
&(objectClass=ipService)
(cn=ldap)(ipServiceProtocol=tcp))
```
 - Or

```
(|(cn= Tweety)(cn=Road Runner))
```
 - Not

```
(!(cn= Tweety))
```

Slide 24

Extensible Filters

- Extensible filters allow us to change the way a match is performed

- Change how value comparison is done

```
(cn:2.5.13.5:=Road Runner)
```

```
(cn:caseExactMatch:=Road Runner)
```

- Filters only match the attributes of an entry, not the DN components, unless specified

```
(ou:dn:=bird)
```

```
(ou:dn:caseIgnoreMatch:=coyote)
```

Slide 25

Performing a search

```
$r = $ldap->search(  
    base    => 'dc=cartoon,dc=com',  
    scope  => 'subtree',  
    filter => '(cn=Road Runner)'  
);
```

```
die $r->error if $r->code;
```

- The result object also contains the matching entries

```
foreach my $entry ($r->entries) {  
    process_entry( $entry );  
}
```

Slide 26

The Entry object

- Used for both creating new entries and in retrieval of existing objects

- dn

- Returns the DN for the entry

```
$DN = $entry->dn;
```

- exists

- Test if an attribute exists within the entry

```
do_something() if $entry->exists('cn');
```

Slide 27

The Entry object - cont.

- get_value

- Obtain the value for an attribute in the entry

```
$value = $entry->get_value('cn');
```

- For multi-valued attributes get_value will return the first value in a scalar context and all of them in a list context

```
$first = $entry->get_value('objectClass');
```

```
@values = $entry->get_value('objectClass');
```

```
$values = $entry->get_value('objectClass',  
                           asref => 1);
```

Slide 28

The Entry object - cont.

- attributes
 - Return a list of attribute names that the entry contains
- ```
@attrs = $entry->attributes;
```
- NOTE: Attribute names should be treated as case-insensitive

Slide 29

## Displaying an entry

- If you know that all values in your entry are printable, the following could be used to display the entry contents

```
sub display_entry {
 my $entry = shift;

 my @attrs = $entry->attributes;

 foreach my $attr (@attrs) {
 my @value = $entry->get_value($attr);

 foreach my $value (@value) {
 print "$attr: $value\n";
 }
 }
}
```

Slide 30

## Controlling what's returned

- Obtaining just attribute names
  - By default an LDAP server will return the attributes and their values for each entry
  - Asking for only the types, the server will return the same entries as before, but the value for each attribute will be an empty list

```
$r = $ldap->search(
 base => 'dc=cartoon,dc=com',
 filter => '(cn=Road*)',
 typesonly => 1
);
```

Slide 31

## Controlling what's returned

- Which attributes are returned depend on your permissions. You can override this by giving a list of attributes

```
$r = $ldap->search(
 base => 'dc=cartoon, dc=com',
 filter => '(cn=Road*)'
 attrs => [qw(cn)]
);
```

- This can be combined with types only to determine if entries have certain attributes

Slide 32

## Referrals

- A referral is returned by the server when the whole request needs to be resent to a different server
- A referral can be returned in response to any operation, except unbind and abandon
- Detected by a result code of LDAP\_REFERRAL

```
$r = $ldap->search(...);
@ref = $r->referrals if $r->code == LDAP_REFERRAL;
```
- Each referral is an LDAP URL
- Net::LDAP does not provide the option to automatically follow referrals

Slide 33

## References

- A reference is returned by the server when part of the request must be sent to a different server
- A reference can only be returned by a search operation
- There is no result code to detect these

```
$r = $ldap->search(...);
@ref = $r->references;
```
- Each reference is an LDAP URL
- Net::LDAP does not provide the option to automatically follow references

Slide 34

## Adding

- There are four different ways that Net::LDAP supports adding new entries into a directory
  - The add method
  - The Entry class
  - LDIF
    - Same as adding with the Entry class, except the Entry is read from a file via the LDIF module
  - DSML
    - Same as adding with the Entry class, except the Entry is read from a file via the DSML module

Slide 35

## Adding – the add method

- Data structure of entry is passed directly to the add method

```
$DN = 'cn=Road Runner, ou=bird, dc=cartoon, dc=com';
$r = $ldap->add($DN,
 attrs => [
 cn => 'Road Runner',
 objectClass => 'cartoonCharacter',
 food => 'Bird Seed'
]
);
```

Slide 36

## Adding – the Entry class

- An Entry is built and passed to the add method

```
$DN = 'cn=Road Runner, ou=bird, dc=cartoon, dc=com';
$e = Net::LDAP::Entry->new($DN);
$e->add(cn => 'Road Runner');
$e->add(
 objectClass => 'cartoonCharacter',
 food => 'Bird Seed'
);
$r = $ldap->add($e);
```

Slide 37

## Deleting

- To delete an Entry you need to know the DN of the entry to be deleted

```
$DN = 'cn=Road Runner, ou=bird, dc=cartoon, dc=com';
$ldap->delete($DN);
```

- Alternatively, like many Net::LDAP methods, an Entry can be passed where a DN is expected

```
$entry = find_entry_to_delete();
$ldap->delete($entry);
```

Slide 38

## Modifying

- The modify operation has three sub-operations
  - Add
    - Add new attributes
    - Add values to existing multi-valued attributes
  - Delete
    - Delete whole attributes
    - Delete values from within existing attributes
  - Replace
    - Replace existing attributes, or add if necessary

Slide 39

## Modify – add

- To add a new attribute to an entry, or value to an existing attribute

```
$r = $ldap->modify($DN,
 add => { mail => 'rr@cartoon.com' }
);
```

- An error will be returned if
  - If the attribute exists and is not multi-valued
  - If the attribute exists and is multi-valued and the value already exists
  - The schema does not allow the attribute to be added

Slide 40

## Modify – delete

- To delete whole attributes

```
$r = $ldap->modify($DN,
 delete => ['mail']
);
```

- Or, specific values from an attribute

```
$r = $ldap->modify($DN,
 delete => { mail => ['tweety@cartoon.com'] }
);
```

- If deleting specific values from an attribute leaves the attribute with an empty list, the attribute is deleted
- If the given attribute does not exist, the error code LDAP\_NO\_SUCH\_ATTRIBUTE is returned

Slide 41

## Modify – replace

- Replace allows you to replace whole attributes

- Single value

```
$r = $ldap->modify($DN,
 replace => { mail => 'bird@cartoon.com' }
);
```

- Multi-valued

```
$r = $ldap->modify($DN,
 replace => {
 mail => ['bird@cartoon.com', 'bird@acme.com']
 }
);
```

- Delete

```
$r = $ldap->modify($DN, replace => { mail => [] });
```

Slide 42

## Renaming an Entry

- `moddn` allows an entry to be moved within the tree

```
$DN = 'cn=Road Runner, ou=bird, dc=cartoon, dc=com';
$r = $ldap->moddn($DN,
 newrdn => 'cn=Lunch'
);
```

- An entry can also be moved to another branch of the tree

```
$DN = 'cn=Wile E. Coyote, ou=coyote, dc=cartoon, dc=com';
$r = $ldap->moddn($DN,
 newrdn => 'cn=Isac Newton',
 newsuperior => 'ou=genius, dc=cartoon, dc=com'
);
```

- An error will be returned if the new entry already exists

Slide 43

## Comparing

- `Compare` allows the value of an attribute to be compared with a value following the comparison rules specified in the directory schema

```
$DN = 'cn=Thieves Hideout, ou=cave, dc=cartoon, dc=com';
$r = $ldap->compare($DN,
 attr => 'userPassword',
 value => 'opensesame'
);
```

- On success `$r->code` will be either `LDAP_COMPARE_TRUE` or `LDAP_COMPARE_FALSE`
- Some systems impose access rules on some attributes that allow them to be compared but not read, eg `userPassword`

Slide 44

## Groups

- In a directory a group is implemented as an entry with a multi-valued attribute
- This attribute is normally called member
- The contents of the attribute are the DNs of the members

```
$DN = "cn=Animaniacs, ou=Groups, dc=cartoon, dc=com";
$r = $ldap->modify($DN,
 add => {
 member => [
 "cn=Wakko, ou=character, dc=cartoon, dc=com",
 "cn=Yakko, ou=character, dc=cartoon, dc=com",
 "cn=Dot, ou=character, dc=cartoon, dc=com"
]
 }
);
```

Slide 45

## LDIF

- LDIF is a text format used to represent entries within a directory

```
dn: cn=Road Runner, ou=bird, dc=cartoon, dc=com
objectClass: cartoonCharacter
cn: Road Runner
food: Bird Seed
```

- It can also be used as a command language to manipulate existing directory entries

```
dn: cn=Road Runner, ou=bird, dc=cartoon, dc=com
changetype: modify
replace: mail
mail: road.runner@cartoon.com
```

Slide 46

## Reading LDIF

- LDIF files can be read using the Net::LDAP::LDIF class

```
$ldif = Net::LDAP::LDIF->new($filename, "r");
while ($entry = $ldif->read_entry) {
 $r = $ldif->add($entry);
 warn $entry->dn,": ",$r->error if $r->code;
}
warn "Error reading $filename" unless $ldif->eof;
```

- The \$filename passed to ->new may alternatively be a file handle

```
$ldif = Net::LDAP::LDIF->new(*STDIN, "r");
```

Slide 47

## Writing LDIF

- The same class can be used to write LDIF

```
$ldif = Net::LDAP::LDIF->new($filename, "w");
$ldif->write_entry($entry);
```

- Appending can be done by passing "a" instead of "w"

Slide 48

## Controls

- Version 3 made the protocol extendable by adding controls
- Most Net::LDAP methods accept a list of controls. This is passed as an array reference using the `control` named parameter
- Net::LDAP currently implements
  - Paged Search Results
  - Sorted Search Results
  - Virtual List View
  - Proxy Authentication

Slide 49

## Paged Search Results

- Allows the results of a search to be returned in blocks instead of all at once

```
$page = Net::LDAP::Control::Paged->new(size => 5);
@args = (
 base => "DC=cartoon, DC=com",
 scope => "subtree",
 filter => "(cn=road*)",
 control => [$page]
);
while(1) {
 $r = $ldap->search(@args);
 $r->code and last;
 $ctrl = $mesg->control(LDAP_CONTROL_PAGED)
 or last;
 $cookie = $ctrl->cookie or last;
 $page->cookie($cookie);
 process_entries($r) or last;
}
```

Slide 50

## Paged Search Results – cont.

```
If we terminated early tell the server we are finished

if ($cookie) {
 $page->cookie($cookie);
 $page->size(0);
 $ldap->search(@args);
}
```

Slide 51

## Sorted Search Results

- Controls the order of the entries returned from a search

```
$sort = Net::LDAP::Control::Sort->new(
 order => "cn -phone"
);

$r = $ldap->search(@args, control => [$sort]);

($ctrl) = $mesg->control(LDAP_CONTROL_SORTRESULT);

print "Results are sorted\n"
 if $ctrl and !$ctrl->result;

@entries = $r->entries;
```

Slide 52

## Virtual List Views

- Controls a search to only return a subset of the matching entries using a moving window
- A sort control must be passed with the VLV control

```
$vlv = Net::LDAP::Control::VLV->new(
 offset => 1, # Target entry is the first
 before => 0, # No entries from before target entry
 after => 19, # 19 entries after target entry
);

$sort = Net::LDAP::Control::Sort->new(sort => 'cn');
```

Slide 53

## Virtual List View – cont

```
@args = (
 base => "dc=cartoon, dc=com",
 scope => "subtree",
 filter => "(objectClass=*)",
 control => [$vlv, $sort]
);

$r = $ldap->search(@args);
($ctrl) = $mesg->control(LDAP_CONTROL_VLVRESPONSE)
 or die;
$vlv->response($ctrl);

$vlv->end;
$r = $ldap->search(@args);
($ctrl) = $mesg->control(LDAP_CONTROL_VLVRESPONSE)
 or die;
$vlv->response($ctrl);
```

Slide 54

## Virtual List View – cont

- Other VLV methods include

- Move to the start of the list

```
$vlv->start;
```

- Scroll through the list

```
$vlv->scroll_page(1);
$vlv->scroll_page(-1);
```

- Move to a particular entry

```
$vlv->assert("B");
```

- Query

```
$offset = $vlv->offset;
$content = $vlv->content;
$before = $vlv->before;
$after = $vlv->after;
```

Slide 55

## Proxy Authentication

- Allows a bound client to assume the identity of another entity
- Requires access control permission from the assumed identity

```
$proxyDN =
 "cn=Friz Freleng, ou=animator, dc=cartoon, dc=com";
$ctrl = Net::LDAP::Control::ProxyAuth->new(
 proxyDN => $proxyDN
);
$DN = "cn=Sylvester, ou=cat, dc=cartoon, dc=com";
$r = $ldap->modify($DN,
 delete => { food => 'birds' }
);
```

Slide 56

## Schema

- The schema for a directory is stored in an entry within the directory
- Net::LDAP provides a single method for obtaining the directory schema

```
$schema = $ldap->schema;
```

- The result is a Net::LDAP::Schema object
- Net::LDAP::Schema provides an API for querying the content of the schema
- Net::LDAP does not provide a method to modify schema

Slide 57

## Handling Errors

- The Message object returned by most methods contains the status of the operation performed
- The code method will return a status code. For most operations a success will be represented as a zero in this field
  - Net::LDAP::Constant exports LDAP\_SUCCESS as zero
- The error method will return a short error message. This may be either a message actually returned by the server, or a default message that corresponds to the code returned by the server

Slide 58

## Handling Errors – cont

- In most simple cases the result code should be LDAP\_SUCCESS, or in the case of compare LDAP\_COMPARE\_FALSE or LDAP\_COMPARE\_TRUE
- Net::LDAP can be told to call a subroutine whenever an unexpected result is returned
- The following will cause Net::LDAP to die when an unexpected result is returned

```
$ldap = Net::LDAP->new($server, onerror => 'die');
```

Slide 59

## Handling Errors – cont

- Other handlers are
  - 'warn' – Net::LDAP will call warn with an error messages, but will continue as normal
  - 'undef' – Net::LDAP will call warn with an error if -w is in effect. The method called will then return undef instead of the usual message object
- onerror may also be passed a sub reference. This sub will be called with the message object. The return value from this sub is what will be returned by the method that was called

Slide 60

## Net::LDAP::Util

- The Net::LDAP::Util module contains many utility functions
  - `ldap_error_name` - Given an error code it will return the name associated with that code

```
warn ldap_error_name(0); # "LDAP_SUCCESS"
```
  - `ldap_error_message` - Given an error code it will return a short error message

```
warn ldap_error_message(0); # "Success"
```
  - `ldap_error_text` - Given an error code it will return the text from the Net::LDAP::Constant POD for the given error

Slide 61

## Net::LDAP::Util

- Distinguished Names can be represented in many forms, so comparing two DN's is not as simple as using eq. For this reason Net::LDAP::Util provides `canonical_dn`
- `canonical_dn` performs the following actions on a DN
  - Downcase any hexadecimal codes
  - Uppercase all type names
  - Remove the letters OID from any OID
  - Escape all special characters
  - Escapes all leading and trailing spaces
  - Orders the parts within any multi-part RDN

Slide 62

## Using callbacks

- LDAP returns results to the client in multiple packets. Each packet will contain a single entry, a reference or a result code
- Net::LDAP accumulates all packets for a transaction before a method returns
- Callbacks allow the entries to be processed as they arrive, instead of waiting for the completion
- This can result in less memory consumption and also faster processing

Slide 63

## Using callbacks – cont

```
$r = $ldap->search($DN,
 filter => '(objectClass=*)',
 callback => \&process
);
```

- process will be called for each packet received
- The first argument passed will be the message object
- If the packet was an entry, then there will be a second argument, which will be a Net::LDAP::Entry object
- If the packet was a reference, there will be a second argument, which will be a Net::LDAP::Reference object

Slide 64

## Using callbacks – cont

```
sub process {
 my ($r, $obj) = @_ ;

 if (!$obj) {
 print $r->error, "\n";
 }
 elsif ($obj->isa('Net::LDAP::Entry')) {
 print $obj->dn, "\n";
 $r->pop_entry;
 }
 else {
 foreach my $ref ($obj->references) {
 print $ref, "\n";
 }
 }
}
```

Slide 65

## LDAP and XML

- Directory Service Markup Language (DSML) is the XML standard for representing directory service information in XML
- DSML is still very new
- DSML cannot be used as a command language like LDIF

Slide 66

## Client Software

---

---

- perl-ldap
  - <http://perl-ldap.sourceforge.net/>
- PerLDAP
  - <http://www.mozilla.org/directory/perldap.html>
- Netscape
  - <http://www.iplanet.com/>
- Sun JNDI
  - <http://www.java.sun.com/jndi/>

Slide 67

## Open Source Servers

---

---

- OpenLDAP
  - <http://www.openldap.org/>
- JavaLDAP
  - <http://javaldap.sourceforge.net/>

Slide 68

## Commercial Servers

---

---

- Iplanet
  - <http://www.iplanet.com/>
- Messaging Direct
  - <http://www.messagingdirect.com/>
- Nexor
  - <http://www.nexor.com/>
- Critical Path
  - <http://www.cp.net/>

Slide 69

## perl-ldap resources

---

---

- Homepage
  - <http://perl-ldap.sourceforge.net/>
- Online documentation
  - <http://perl-ldap.sourceforge.net/doc/>
- FAQ
  - <http://perl-ldap.sourceforge.net/FAQ.html>
- Mailing list
  - [perl-ldap-dev@lists.sourceforge.net](mailto:perl-ldap-dev@lists.sourceforge.net)

Slide 70