Extracted from:

Enterprise Integration with Ruby

A Pragmatic Guide

has many useful features (such as table inheritance) that we did not cover here. Additionally, its capabilities differ from database system, to database system so, for example, not all the features that are available with MySQL are also available if you use DB2.

2.4 Lightweight Directory Access Protocol (LDAP)

We use directories in the real world all the time: telephone books, lists of network accounts, address books, the domain name service (DNS), and so on. Typically, directories are organized hierarchically—as trees—and their entries are often read and rarely modified.

Implementing directories with relational database systems can be a bit complicated. Even though many database vendors added tools for hierarchical queries to their products, using them is still far from being convenient. (Some vendors, including Oracle, even ship a separate directory service that is based on their relational database product.)

Because of this, a standard for accessing directories was created as part of the *X.500 directory specification*. It was called *Directory Access Protocol* (DAP). Unfortunately, it was both complex and complicated, and no one implemented it completely.

X.500 directory specification

Directory Access Protocol

As a consequence, an easier standard was defined: the Lightweight Directory Access Protocol (LDAP).[17] This is the most widespread directory service in use today.

I'll give a short introduction to LDAP in the rest of this section. If you're already familiar with LDAP you can safely skip it and go directly to Section 2.4, *An Address Book for PragBouquet Customers*, on page 55.

Simply put, LDAP is to directories what SQL is to relational databases. It helps you to model real-world entities as *directory entries* (not as tables) that have different attributes. Attributes have a name, a type, and a multidimensional value; i.e., attributes can have a list of values. Every directory entry (from now on we call them *entries* for short) has at least one attribute called objectclass that determines which attributes the entry has.

directory entries

In LDAP you put all object classes and their according attribute type definitions belonging to a particular problem domain into a *schema*.

schema

[17] http://www.faqs.org/rfcs/rfc2251.html

The core schema, for example, contains the definition of the residentialPerson object class:

```
objectclass (
  2.5.6.10
  NAME 'residentialPerson'
  DESC 'RFC2256: an residential person'
  SUP person
  STRUCTURAL
  MUST l
  MAY (
    businessCategory $ x121Address $ registeredAddress $
    destinationIndicator $ preferredDeliveryMethod $
    telexNumber $ teletexTerminalIdentifier $ telephoneNumber $
    internationaliSDNNumber $ facsimileTelephoneNumber $
    preferredDeliveryMethod $ street $ postOfficeBox $
    postalCode $ postalAddress $ physicalDeliveryOfficeName $
    st $ l
  )
)
```

This looks similar to SQL's create table statement, doesn't it? The biggest difference is that the type of the attributes (SQL calls them *columns*) are defined separately. The meaning of the different declarations and keywords is as follows:

- In LDAP, every definition begins with an *object identifier* (OID) that uniquely identifies the object class or attribute type worldwide. OIDs are numbers separated by periods and have to be registered at the *Internet Assigned Numbers Authority* (IANA).[18] Private OIDs always start with 1.3.6.1.4. *object identifier*
- Object classes have a name that is defined with the NAME keyword. To prevent name clashes, you should add a unique prefix or postfix to your own object class and attribute type names.
- DESC lets you give a human-readable description of the object class.
- The SUP keyword points to the superclass of an object class. LDAP is object oriented, and an object class can inherit the attributes of another class. Every class has at least one superclass called top.
- An LDAP class can be a STRUCTURAL, AUXILIARY, or ABSTRACT class. Abstract classes are classes that are meant only to be base classes (such as top). Classes meant to define completely new object hier-

[18] http://www.iana.org/cgi-bin/enterprise.pl

archies are declared as STRUCTURAL. AUXILIARY classes let you "mixin" attributes into existing structural classes.

- MUST expects a dollar-separated list which contains the classes' mandatory attributes.
- MAY expects a dollar-separated list containing the classes' optional attributes.

Attribute types, such as the telephoneNumber attribute we have used in the residentialPerson object class, are defined as follows:

```
attributetype (
  2.5.4.20
  NAME 'telephoneNumber'
  DESC 'RFC2256: Telephone Number'
  EQUALITY telephoneNumberMatch
  SUBSTR telephoneNumberSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50{32}
)
```

Like an object class, the attribute type definition starts with an OID. NAME and DESC have the same meaning as in the object class definition. The remaining keywords have the following meaning:[19]

- EQUALITY specifies which algorithm should be used to test whether two telephoneNumber attributes are equal. This is a little bit more sophisticated than a simple string comparison, because telephone numbers often contain characters only for better readability. For example, "0049 (0) 1234 / 56 78" and "004912345678" are completely different strings, but they represent the same telephone number. The LDAP standard defines a lot of equality algorithms.
- SUBSTR lets you define which algorithm should be used to check whether a particular telephoneNumber number attribute contains a particular substring.
- The SYNTAX element refers to the OID of the attributes' syntax. LDAP defines a syntax for many types that are used often such as integers, strings, timestamps, and even JPEG files.

It's not difficult to build your own object classes and attribute types, but it's certainly a good idea first to check whether the object class you need has not already be defined. LDAP specifies dozens of base classes for

[19] To learn about attribute types, you have to read RFC 2252: http://www.faqs.org/rfcs/rfc2252.html.

all the elements you typically find in directories: person, residentialPerson, organizationalPerson, and so on. Often it's sufficient to derive a new class from an existing one, adding just a few attributes. For example, if you need to store address data containing the geographical position of the address, you can derive a new geoPerson class from residentialPerson, adding longitude and latitude attributes.

That's all not too different from what you do with relational databases (except for the inheritance features), and you could use LDAP to store nonhierarchical data. But usually LDAP repositories represent hierarchical trees of entries belonging to one or more object classes.

distinguished name

relative distinguished names

Each entry has a unique name, the *distinguished name* (DN). The DN consists of several *relative distinguished names* (RDN). An RDN is a list of attribute name/value pairs that are separated by a comma or a semicolon. For example, telephoneNumber=004912345678 could be an RDN with the attribute name telephoneNumber and the value 004912345678. A more precise RDN could be

```
cn=Maik Schmidt,telephoneNumber=004912345678
```

This additionally specifies the cn ("common name") attribute of a person object.

root entry

As we all know, a picture is worth approximately 2^{10} words, so let's have a look at Figure 2.3, on page 59. The *root entry* of the directory in this figure has a DN consisting of two RDNs: dc=pragbouquet,dc=com.[20] It automatically becomes an RDN for all entries in the tree. The deeper you go down the hierarchy, the longer the DNs and RDNs get. For example, the distinguished names of all entries on the left side contain the relative distinguished name uid=4711,dc=pragbouquet,dc=com. Simply put, DNs specify leaves, and RDNs specify subtrees.

LDAP allows you to read, modify, and delete subtrees and single nodes of your directories. In relational databases you specify particular rows with a WHERE clause in your SQL statements. In LDAP you use RDNs and DNs to do so.

LDAP Data Interchange Format

We mentioned before that directory entries are often read and rarely updated. Hence, the LDAP standard defined a technology that makes an initial import of directory entries easy: the *LDAP Data Interchange Format* (LDIF).[21] It's a simple textual file format for describing directory

[20] dc stands for *domain component*. dc is a mandatory attribute for entries belonging to the organization object class.

entries. Here's an LDIF representation of the root entry and one of its descendants of our sample directory:

```
# First (root) entry: the PragBouquet organization.
dn: dc=pragbouquet,dc=com
objectclass: dcObject
objectclass: organization
o: PragBouquet
dc: pragbouquet

# Second entry: an address book for customer 4711.
dn:uid=4711,dc=pragbouquet,dc=com
objectclass: top
objectclass: person
objectclass: uidObject
uid: 4711
cn: John Jackson
sn: Jackson
description: Address book of John Jackson.
```

LDIF is line oriented. Comment lines start with a # character. All the other lines represent an attribute and its corresponding value, separated by a colon. If an attribute has more than one value, it may appear several times. Every LDAP server comes with a bunch of utilities for modifying an existing repository and for importing .ldif files.

Although a lot of directory services work more or less invisibly, touched only by your system administrators, chances are good that you'll have to integrate with one someday, because LDAP is gaining popularity among application developers, too. In the following sections we'll show how to manipulate a directory service based on OpenLDAP with Ruby.

An Address Book for PragBouquet Customers

The marketing department made yet another astonishing observation: there are people who celebrate their birthdays every year! Wouldn't it be great if PragBouquet customers could easily send them a bouquet on those birthdays? And wouldn't it be nice if PragBouquet customers could be spared the extra work of entering the same address data for the recipients, over and over again?

So, marketing came up with an ingenious idea. All PragBouquet customers should have their own address book where they can store the addresses of the people they've ever sent a bunch of flowers.

[21] http://www.faqs.org/rfcs/rfc2849.html

The web shop team said that it's not a big deal to create a user interface for the address book, but they asked you to create the corresponding backend services. Fortunately, they want to give Ruby on Rails[22] a try, so you can use Ruby for implementing the address book logic.

When thinking about things like address books, LDAP immediately comes to mind, so you decide to implement the address book as a directory service using the OpenLDAP[23] system. It has everything you need, it's available for free, it works on top of several database systems, and it ships with several utilities for reading and manipulating data.

For the development phase we install an OpenLDAP server on our local machine and configure it using this configuration file:

File 39

```
include /sw/etc/openldap/schema/core.schema

database bdb
suffix "dc=pragbouquet,dc=com"
rootdn "cn=root,dc=pragbouquet,dc=com"
rootpw secret
directory /sw/var/openldap-data
index objectclass eq
```

That is really all we need to get our address book application up and running. We have to include the core schema, because we'll need some of its definitions (person, residentialPerson, and uidObject). In addition, we have to define the database we want to use (the LDAP standard does not define how the directory is to be stored). It's a Berkeley DB (bdb)[24] with all data files stored in directory /sw/var/openldap-data. The distinguished name of our root node (needed for administrative purposes only) is cn=root,dc=pragbouquet,dc=com. We have to authenticate ourselves using the nearly unbreakable plain-text password *secret* whenever we want to write to the database.

LDAP allows you to create a sophisticated directory layout for address books comprising lots of organizational units or even define your own object classes, but we will use a more modern and simpler approach. We will organize our directory in a flat way using domain components and uid attributes.[25]

[22] http://www.rubyonrails.com
[23] http://www.openldap.org
[24] http://sleepycat.com
[25] http://www.faqs.org/rfcs/rfc2377.html

Before diving into Ruby code, let's take a closer look at the directory structure and then initialize our repository with some sample data stored in init.ldif:

File 34

```
# Create the PragBouquet organization.
dn: dc=pragbouquet,dc=com
objectclass: dcObject
objectclass: organization
o: PragBouquet
dc: pragbouquet

# Create an address book for customer 4711.
dn:uid=4711,dc=pragbouquet,dc=com
objectclass: top
objectclass: person
objectclass: uidObject
uid: 4711
cn: John Jackson
sn: Jackson
description: Address book of John Jackson.

# Create the first address book entry for customer 4711.
dn:cn=Marge Jackson,uid=4711,dc=pragbouquet,dc=com
objectclass: top
objectclass: residentialPerson
cn: Marge Jackson
sn: Jackson
l: Springfield
st: IL
street: Evergreen Terrace 42
postalCode: 62701
description: Don't forget our wedding anniversary!

# Create the second address book entry for customer 4711.
dn:cn=P.H. Beans,uid=4711,dc=pragbouquet,dc=com
objectclass: top
objectclass: residentialPerson
cn: P.H. Beans
sn: Beans
l: Springfield
st: MO
street: Nuclear Powerplant Road 1
postalCode: 65801
description: My boss.

# Create an address book for customer 0815.
dn:uid=0815,dc=pragbouquet,dc=com
objectclass: top
objectclass: person
objectclass: uidObject
uid: 0815
```

```
cn: Max Mustermann
sn: Mustermann
description: Address book of Max Mustermann.

# Create the first address book entry for customer 0815.
dn:cn=Jane Doe,uid=0815,dc=pragbouquet,dc=com
objectclass: top
objectclass: residentialPerson
cn: Jane Doe
sn: Doe
street: 125 N. Arbitrary Street
st: DC
l: Washington
postalCode: 20500
description: My Sweetheart!
```

The previous LDIF file should be nearly self-explanatory (comment lines start with a # character). Every entry has a *distinguished name* (DN). All its other attributes are listed as "key: value" pairs. All attributes are potentially multidimensional, so they may appear several times.

distinguished name

Note that we use the attribute uid to structure our address books. Every web shop user is identified by a particular identifier (it might be a customer ID, an e-mail address, or something similar). Whenever a customer creates a completely new address book (not an address book entry), a new directory entry for her user ID will be added. The directory belonging to our init.ldif file looks like Figure 2.3, on the next page (we have left out most attributes for brevity).

Let's start our server and load the initial data using the ldapadd command:

```
mschmidt:~/ldap> sudo slapd
Password:
mschmidt:~/ldap> ldapadd -c -x -D "cn=root,dc=pragbouquet,dc=com" \
> -W -f init.ldif
Enter LDAP Password:
adding new entry "dc=pragbouquet,dc=com"

adding new entry "uid=4711,dc=pragbouquet,dc=com"

adding new entry "cn=Marge Jackson,uid=4711,dc=pragbouquet,dc=com"

adding new entry "cn=P.H. Beans,uid=4711,dc=pragbouquet,dc=com"

adding new entry "uid=0815,dc=pragbouquet,dc=com"

adding new entry "cn=Jane Doe,uid=0815,dc=pragbouquet,dc=com"
```

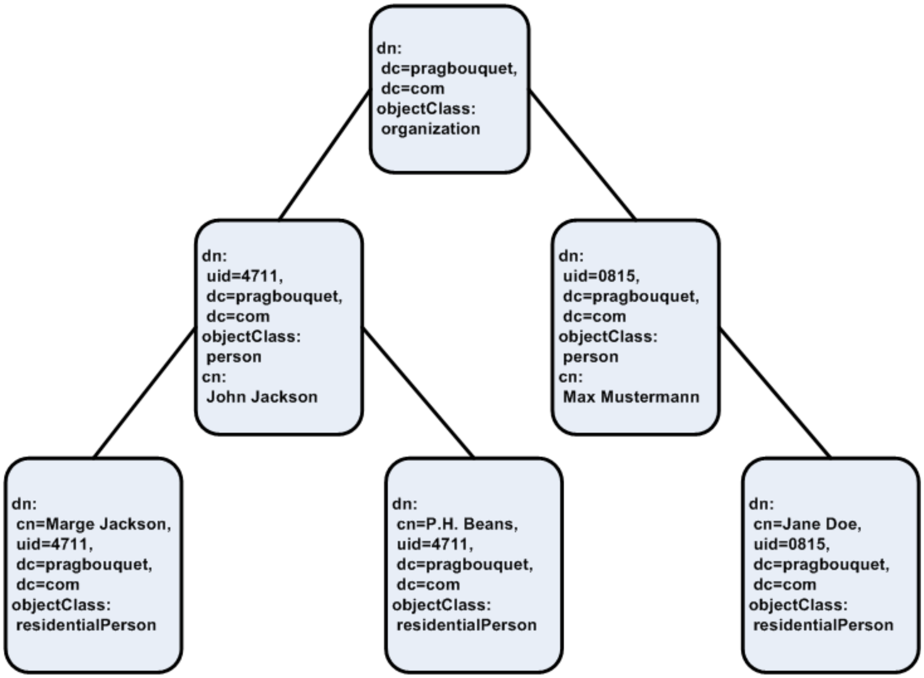

Figure 2.3: Address Book Layout

Our .ldif file didn't contain any errors, and six new entries have been created.

OpenLDAP's ldapsearch command allows us to query the repository. It prints its results in LDIF. To become a bit more familiar with our directory, let's print the address book of the user identified by uid 4711:

```
mschmidt:~/ldap> ldapsearch -x -s one \
> -b 'uid=4711,dc=pragbouquet,dc=com' \
> '(objectclass=*)'
# extended LDIF
#
# LDAPv3
# base <uid=4711,dc=pragbouquet,dc=com> with scope one
# filter: (objectclass=*)
# requesting: ALL
#

# Marge Jackson, 4711, pragbouquet.com
dn: cn=Marge Jackson,uid=4711,dc=pragbouquet,dc=com
objectClass: top
```

```
objectClass: residentialPerson
cn: Marge Jackson
sn: Jackson
l: Springfield
st: IL
street: Evergreen Terrace 42
postalCode: 62701
description: Don't forget our wedding anniversary!

# P.H. Beans, 4711, pragbouquet.com
dn: cn=P.H. Beans,uid=4711,dc=pragbouquet,dc=com
objectClass: top
objectClass: residentialPerson
cn: P.H. Beans
sn: Beans
l: Springfield
st: MO
street: Nuclear Powerplant Road 1
postalCode: 65801
description: My boss.

# search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
```

Obviously, everything is up and running. Our query returned the two address book entries that belong to the customer identified by user ID 4711. But what are those options we passed to the command?

- `-x` uses the simple authentication mechanism. In our case the communication is unencrypted, and no password is needed.
- `-s one` searches the directory "one level beyond base," so it returns all entries below our search base, but not the base itself. `-s base` would have returned the base object only, and `-s sub` would have returned the base object and all its descendants.
- `-b 'uid=4711,dc=pragbouquet,dc=com'` sets the search base to the distinguished name `uid=4711,dc=pragbouquet,dc=com`, so that all entries of the subtree belonging to this DN will be returned.
- `(objectclass=*)` specifies a filter for the entries to be returned. The `(objectclass=*)` filter is comparable to SQL's `SELECT *` statement and selects all entries no matter what attributes they have. If we were interested in entries from Illinois only, we could have set the filter to `(st=IL)`.

In the following sections we'll see how to manipulate our repository with Ruby.

Ruby/LDAP

The Ruby/LDAP[26] library was initially created by Takaaki Tateishi and is currently maintained by Ian Macdonald. It supports all LDAP clients that comply with the LDAP Application Program Interface.[27] You can use Ruby/LDAP to interface with OpenLDAP, Netscape, and ActiveDirectory, among others.

As a first exercise we'll try to read John Jackson's address book. It should not be too surprising that accessing a directory service looks similar to accessing a relational database system:

File 33

```
require 'pp'
require 'ldap'
include LDAP

begin
  connection = Conn.new('127.0.0.1', LDAP_PORT)
  connection.set_option(LDAP_OPT_PROTOCOL_VERSION, 3)
  connection.bind do
    base_dn = 'uid=4711,dc=pragbouquet,dc=com'
    scope = LDAP_SCOPE_ONELEVEL
    filter = '(objectClass=*)'
    connection.search(base_dn, scope, filter) do |entry|
      pp entry.to_hash
    end
  end
rescue Exception => ex
  puts ex
end
```

This prints the following:

```
{"cn"=>["Marge Jackson"],
 "st"=>["IL"],
 "l"=>["Springfield"],
 "sn"=>["Jackson"],
 "description"=>["Don't forget our wedding anniversary!"],
 "postalCode"=>["62701"],
 "street"=>["Evergreen Terrace 42"],
 "objectClass"=>["top", "residentialPerson"],
 "dn"=>["cn=Marge Jackson,uid=4711,dc=pragbouquet,dc=com"]}
{"cn"=>["P.H. Beans"],
```

[26] http://ruby-ldap.sourceforge.net
[27] http://www.faqs.org/rfcs/rfc1823.html

```
"st"=>["MO"],
"l"=>["Springfield"],
"sn"=>["Beans"],
"description"=>["My boss."],
"postalCode"=>["65801"],
"street"=>["Nuclear Powerplant Road 1"],
"objectClass"=>["top", "residentialPerson"],
"dn"=>["cn=P.H. Beans,uid=4711,dc=pragbouquet,dc=com"]}
```

First, we create a new connection to the LDAP service by calling the method LDAP::Conn.new(host='localhost',port=LDAP_PORT). We then set the LDAP_OPT_PROTOCOL_VERSION option, because we've set up an LDAPv3 service (it's OpenLDAP's default).

In line 8 we bind our connection object to the server. The real work is performed in the code block we pass to the bind(dn=nil, password=nil, method=LDAP_AUTH_SIMPLE) method. The heart of our "program logic" is the search() method. It expects the following parameters:

1. base_dn contains the base DN of the subtree to search in.
2. scope defines the search scope; one of: LDAP_SCOPE_ONELEVEL, LDAP_SCOPE_SUBTREE, or LDAP_SCOPE_BASE.

   In our example we have used LDAP_SCOPE_ONELEVEL, which means "one level beyond base." We are not interested in the base object (the address book owner) itself.

   If we had set the scope to LDAP_SCOPE_SUBTREE the program would have printed the entry for the address book owner, too:

   ```
   {"cn"=>["John Jackson"],
    "sn"=>["Jackson"],
    "uid"=>["4711"],
    "description"=>["Address book of John Jackson."],
    "objectClass"=>["top", "person", "uidObject"],
    "dn"=>["uid=4711,dc=pragbouquet,dc=com"]}
    ...
   ```

   LDAP_SCOPE_BASE returns only the base object (the address book owner in our case).
3. filter contains the LDAP search filter to be used.
4. The attributes array contains the name of the attributes which will be returned. If it is empty or nil (the default), all attributes are returned.
5. The attributes_only flag indicates whether only the names of the attributes should be returned (true). When it is set to false (the default), it returns both names and values.

6. seconds specifies the seconds portion of the search timeout. It defaults to 0. If either this parameter or the useconds parameter is greater than 0, the timeout mechanism will be activated.

7. useconds specifies the microseconds portion of the search timeout. It defaults to 0. If this parameter or the seconds parameter is greater than 0, the timeout mechanism will be activated. To set a timeout of 2.5 seconds, set seconds to 2 and useconds to 500.

8. sort_attribute specifies the attribute by which to sort the search result entries. If no sort attribute is specified (the default), the order of the result entries is unpredictable.

9. sort_proc may contain a code block that is used for sorting the entries returned by the server. It defaults to nil, so the order of the result entries is unpredictable.

search() is an iterator. It expects a code block that gets passed the current entry as an LDAP::Entry object. In line 13 we turn these objects into hashes and print them, nicely formatted.

Reading LDAP entries seems to be fairly easy. Let's try to create new ones now. First let's add an empty address book for Jane Doe (she is already a member of Max Mustermann's address book, but that doesn't matter, because for us they are two different customers):

File 32

```
User = Struct.new(:uid, :forename, :surname)
class AddressBook
  BASE_DC = 'dc=pragbouquet,dc=com'

  attr_reader :user

  def initialize(connection, user)
    @connection, @user = connection, user
  end

  def AddressBook.create(connection, user)
    cn = user.forename + ' ' + user.surname
    adr_book = []

    [
      ['objectclass', %w(top person uidObject)],
      ['uid', [user.uid]],
      ['cn', [cn]],
      ['sn', [user.surname]],
      ['description', ['Address book of ' + cn]]
    ].each do |attr, values|
      adr_book << LDAP.mod(LDAP_MOD_ADD, attr, values)
    end
```

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help programmers stay on top of their game.

Visit Us Online

Enterprise Integration with Ruby
pragmaticprogrammer.com/titles/fr_eir
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
pragmaticprogrammer.com/updates
Be notified when updates and new books become available.

Join the Community
pragmaticprogrammer.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
pragmaticprogrammer.com/news
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragmaticprogrammer.com/titles/fr_eir.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragmaticprogrammer.com/catalog
Customer Service:	orders@pragmaticprogrammer.com
Non-English Versions:	translations@pragmaticprogrammer.com
Pragmatic Teaching:	academic@pragmaticprogrammer.com
Author Proposals:	proposals@pragmaticprogrammer.com