CSCI3171: Network Computing
Faun: A Lightweight IMAP Mail Client

— Project —

Alex Safatli

Monday, April 8, 2013

Contents

Problem Overview

Problem Description

The purpose of this project is to implement a Internet Message Access Protocol (**IMAP**) interface to provide functionality that builds on but hides the protocol. The following was desired to be manifested in this interface:

- as many functions as possible would be implemented from the prescribed **IMAP 4rev1** request for comments (**RFC**),
- as little as possible use of outside libraries; most of the functionality will be derived from base socket communications,
- the handling of communication with any SMTP-enabled **mail server** that accepts IMAP messages, and
- secure communication of credentials, preferably through a Secure Sockets Layer (**SSL**).

In order to serve the comforts of the programmer, this would preferably be programmed in **Python**.

Solution

The solution that came about is a Python program designed as a text-based mail retrieval solution in order to contact a specified hostname for a mail server at a specified port number (this would be typically 993 for SSL-communication or 143 for typical IMAP communication).

This program is entitled **Faun**. The origin of the name is from the film *Pan's Labyrinth*, sometimes known by its Spanish name *El Laberinto del Fauno* (the Labyrinth of the Faun). As a Faun is typically a mythical servant, this seemed like a fitting name for a client designed to serve as a mail retrieval client.

Design

Classes and Code Structure

Two files with code in them are present in order to comprise the program that was created in order to represent two different modules, both key to the functionality of this program:

1. `faun.py`: encompasses the working program; will possess flags in order to specify parameters including the hostname of the computer to connect to a port number, and a username. Furthermore, the function of this client will be to ask the user for further instruction after initiating a connection with the mail server.
2. `dispatcher.py`: encompasses all stream/socket communication functions in one module so as to encapsulate and hide this information from other elements of the program. In this regard, it takes control and manages format of messages, what actions to take when messages are received, and handles these in the order in which they are received. Messages are also saved by identifying tag for organization.

The `faun.py` file contains class `mail_box`, which is a small helper class equivalent to a C `struct` in order to manage information about mailboxes, and several functions. The `dispatch.py` file contains classes `dispatcher`, `tag` that manages the identifier tag present in low-level IMAP messages, and `response/request` message class definitions for respective responses and request messages from and to the server.

As it can be shown in **Figure 1**, most of the functions underlined in RFC3501 for the IMAP 4rev1 protocol are covered by the main program; the only ones not covered include the `NOOP`, `SUBSCRIBE` and its corresponding functions, `LSUB`, `STATUS`, `APPEND`, `CHECK`, `UID` and `EXPUNGE` commands. Most of these were implemented for reasons of either possessing on reason to or because there did not seem a practical reason to implement it in a mail client of **Faun**'s nature.

- `NOOP`; a command that does nothing,
- `SUBSCRIBE` and its corresponding functions, including `LSUB` which provide very little in terms of functionality to a user,
- `STATUS`; only allows investigation of other mailboxes outside of a selected one,
- `APPEND`; mail delivery is not possible with solely **IMAP** and therefore there was no need to provide the implementation of this element of the protocol — SMTP is also necessary in order to provide this functionality,
- `CHECK`; an implementation-specific command for housekeeping,
- `UID`; a useful utility command that was not used in this implementation, and
- `EXPUNGE`; permanently deletes all messages flagged as deleted, but deletion is managed when a mailbox is closed.

Function

In order to use the program, run the **main program** by either calling `python faun.py` or `./faun.py` if the file has execution permissions and a bash shell is used, with `.` replaced with the directory of the file or left as is if in the directory where the file is.

The designation of host name and port number can be made through command-line flags. In order to see all possible options, type in the following to bring up a help menu:

```
python faun.py -h
```

Data Structures

For the most part, a dictionary was used to store tag-message pairs within the dispatcher along with strings to denote respective protocol name, version number, path, parameters, and operations (see `dispatcher.py`).

Linked lists, implemented as a built-in list, are used to store the contents of consecutive information messages from the server that denotes longer bodies including mail messages, mailbox lists, and lists of messages.

Analysis

Results

The mail client, **Faun**, that was created for this project performs many simple mail tasks and even allow users to output mail messages to external files. For the most part, this client will allow straight-forward and cursory investigation of a mail server and mail messages on that server where, unless the user wishes it, nothing is disturbed on the server.

Utilizing base-level socket communication, the IMAP protocol was mostly made functional and an interface, through the **Faun** program, is provided in order to mask this low-level string processing.

A sample run of the program is included below.

```
python faun.py -s fcspostoffice.cs.dal.ca
>> Username: safatli
>> Password:

Securely logged in as safatli on hostname fcspostoffice.cs.dal.ca.

What would you like to do?
  0. CREATE a mailbox
  1. DELETE a mailbox
  2. RENAME a mailbox
  3. SELECT a mailbox
  4. EXIT and close connection
>> Choose an option: 3

5 mailboxes were found on the server.
  0. INBOX
  1. Drafts
  2. Junk
  3. Sent
  4. Trash
>> Select a mailbox: 0

32 messages in mailbox.
0 recent messages found.

Mailbox: INBOX; specify what to do next:
  0. LIST all messages
  1. SEE all new messages
  2. SET a flag on a message
  3. SEARCH for a message
  4. FETCH a message
  5. DELETE a message
  6. REMOVE a flag from a message
  7. CLOSE this mailbox
>> Choose an option: 3
>> Search for: car

Showing 10 messages at a time (of 3)...
  5. Subject: CSUGRADS> Career Conversations Starting Wednesday, Jan 12 11:30am
  11. Subject: CSUGRADS> Reminder - Career Conversations today 11:30 Room 430
  16. Subject: Assignment 2

Mailbox: INBOX; specify what to do next:
  0. LIST all messages
  1. SEE all new messages
  2. SET a flag on a message
  3. SEARCH for a message
  4. FETCH a message
  5. DELETE a message
  6. REMOVE a flag from a message
  7. CLOSE this mailbox
>> Choose an option: 4
>> Specify message UID (or b to return): 16

--- Begin message 16 ---
Date: Tue, 1 Feb 2011 00:16:16 -0400
Subject: Assignment 2
From: "Nauzer Kalyaniwalla" <nauzerk@cs.dal.ca>
To: all-cs2112@cs.dal.ca


 Hello All,

 Assignment 2 is available on the course web-page:

 http://www.cs.dal.ca/~nauzerk/discrete

 Please look it over carefully  before the next class.

 Nauzer


Faculty of Computer Science
Dalhousie University
```

```
Halifax NS
B3H-1W5
(902)-494-2841

--- End message 16 ---
Save e-mail to file (y/n)? n

Mailbox: INBOX; specify what to do next:
  0. LIST all messages
  1. SEE all new messages
  2. SET a flag on a message
  3. SEARCH for a message
  4. FETCH a message
  5. DELETE a message
  6. REMOVE a flag from a message
  7. CLOSE this mailbox
>> Choose an option: 7

What would you like to do?
  0. CREATE a mailbox
  1. DELETE a mailbox
  2. RENAME a mailbox
  3. SELECT a mailbox
  4. EXIT and close connection
>> Choose an option: 4
```

Strengths and Weaknesses

All of the protocol elements investigated are strongly represented in a text-based format. However, very little manipulation of messages (including replying to messages, forwarding messages, and composing messages) is possible because of the nature of the IMAP protocol: it has no method or technique that allows for the delivery of messages.

In hindsight, a better way to have implemented this program would have been utilizing a **Graphical User Interface** (GUI) as well as implementing the appropriately functionality of SMTP communication in order to compose messages (this would be paired with the `APPEND` function above in order to append sent messages to an appropriate `sent-mail` mailbox).

The graphical user interface would have included a single column for listing messages in a given mailbox and then another portion of the window to present messages.

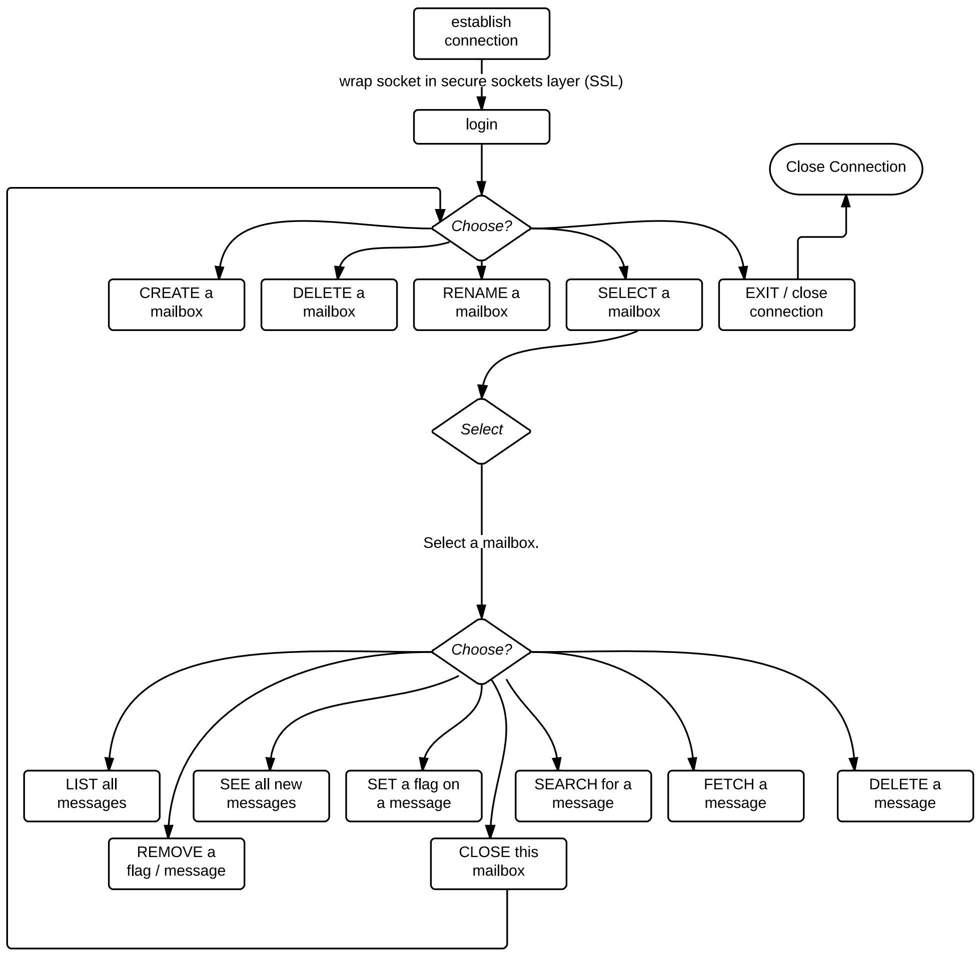

Figure 1: *A small flowchart representing the function of the main program.*