Article
José Pereira · Sep 1, 2021 6m read

Implementing an IMAP Client in InterSystems IRIS - part I

This article explains how to use the InterSystems IRIS platform to write a basic IMAP client. First, we present an overview of IMAP, then we discuss the main IMAP commands and client implementation. Finally, we offer a simple use of this IMAP client on the IRIS interoperability application.

Note that this article isn ' t a detailed IMAP explanation. For more detailed information, please check the references of this article.

IMAP Overview

The Internet Message Access Protocol (IMAP) lets users retrieve emails. Mark Crispin proposed IMAP in the 1980s, and since then, the protocol has been published and revised in RFC 3501. At the time of writing this article, its current version is IMAP4rev1.

It ' s important to note that this protocol is designed for retrieving only. You must use another protocol, the Simple Mail Transfer Protocol (SMTP), if you need to send emails. There ' s also an older protocol for email retrieval that ' s as popular as IMAP, called Post Office Protocol version 3 (POP3).

Trying Basic IMAP Commands

Like POP3, IMAP is a plaintext protocol. You can easily try some of its commands by yourself, using a TCP client like Telnet or OpenSSL.

First, you need your IMAP host and email server port. For instance, at the time of writing this article, this is the host and port to connect to the Microsoft Outlook service:

outlook.office365.com:993

Let ' s connect to this server using our TCP client. Enter the command below and hit ENTER. Note the use of the -crlf flag. This flag is mandatory for IMAP as carriage return and line feed (CRLF) is its line terminator.

```
$ openssl s_client -connect outlook.office365.com:993 -crlf -quiet
```

After you hit enter, the IMAP server shows you some information and stays waiting for input.

```
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert ...
* OK The Microsoft Exchange IMAP4 service is ready. [...]
```

Now we can perform our first IMAP command: CAPACITY. As its name suggests, this command presents features that the server can provide. Note that the first line is the IMAP command itself, and the

others lines are its output - this is the same for all other IMAP commands presented here.

```
TAG1 CAPABILITY
* CAPABILITY IMAP4 IMAP4rev1 AUTH=PLAIN AUTH=XOAUTH2 SASL-
IR UIDPLUS ID UNSELECT CHILDREN IDLE NAMESPACE LITERAL+
TAG1 OK CAPABILITY completed.
```

Note the token before the CAPABILITY command: TAG1. A tag must precede every IMAP command and is any string you want. They identify what command instance the server is responding to. Generally, a simple counter with a prefix fits all your needs.

The IMAP server finishes the response with the tag you used to issue the command and a flag indicating the command ' s status, followed by miscellaneous information related to the command and status. The statuses allowed are: OK (success), NO (failure), and BAD (wrong command).

The command LOGIN expects user and password as arguments. First, let ' s check how the server responds if someone gives an invalid username and password:

```
TAG2 LOGIN user@outlook.com wrong-password
TAG2 NO LOGIN failed.
```

Note the NO status after the command tag.

Now, let ' s proceed to a valid login:

```
TAG3 LOGIN user@outlook.com password
TAG3 OK LOGIN completed.
```

Now we ' re logged in. To finish our session, we use the LOGOUT command:

```
TAG4 LOGOUT
* BYE Microsoft Exchange Server IMAP4 server signing off.
TAG4 OK LOGOUT completed.
```

We need other commands to implement our IMAP client, so we need to reconnect and log into our IMAP server again.

Let ' s get some information about the inbox next.

First, we must tell the server which mailbox we want to access, INBOX in this case.

```
TAG5 SELECT "INBOX"
* 14 EXISTS
* 0 RECENT
...
TAG5 OK [READ-WRITE] SELECT completed.
```

Now we are ready to access messages in our inbox, but we need a way to refer to them first. There are two ways to do that: by message number or by a unique identifier (UID).

Message numbers are just counters, starting from 1 and going up to the number of messages within the inbox. When you delete a message, all following messages have their numbers decremented by 1. You can think of this as an array that has one of its elements removed.

UIDs, on the other hand, keep their value whatever happens to the other messages.

You can get UIDs using the UID SEARCH command. This command accepts a message number if you'd like to get a specific UID or the ALL parameter to get all UIDs in the directory.

In the example below, we search for UID for message number 1, which is 483.

```
TAG6 UID SEARCH 1
* SEARCH 483
TAG6 OK SEARCH completed.
```

Now let ' s retrieve message information, like headers, body, and attachments. We use the FETCH command for this. This command has plenty of parameters, which you can explore in detail on RFC 3501. In this article, we only address parameters related to the needs of the IMAP client implementation.

The first information we need for this demonstration is the message size. We can get this information using the RFC822.SIZE parameter:

```
TAG7 FETCH 1 RFC822.SIZE
* 1 FETCH (RFC822.SIZE 70988)
TAG7 OK FETCH completed.
```

This indicates that the size of message number 1 is 70,988 bytes.

Note that it ' s also possible to use UIDs rather than message numbers to fetch message information:

```
TAG8 UID FETCH 483 RFC822.SIZE
* 1 FETCH (RFC822.SIZE 70988 UID 483)
TAG8 OK FETCH completed.
```

You can also get the basic From, To, Date, and Subject message headers:

```
TAG9 FETCH 1 (FLAGS BODY[HEADER.FIELDS (FROM TO DATE SUBJECT)])
* 1 FETCH (FLAGS (\Seen) BODY[HEADER.FIELDS (FROM TO DATE SUBJECT)] {157}
Date: Thu, 22 Apr 2021 15:49:05 +0000
From: Another User <anotheruser@outlook.com>
To: user@outlook.com
Subject: Greetings from Another User!
 FLAGS (\Seen))
TAG9 OK FETCH completed.
```

Now let ' s retrieve the message body. We can retrieve all the body content using this command:

```
TAG10 FETCH 1 BODY[]
```

```
* 1 FETCH (BODY[] {9599}
...
MIME-Version: 1.0
--00000000000041bd3405c3403048
Content-Type: multipart/alternative; boundary="00000000000041bd3205c3403046"
--00000000000041bd3205c3403046
Content-Type: text/plain; charset="UTF-8"
...
--00000000000041bd3405c3403048
Content-Type: image/png; name="download.png"
Content-Disposition: attachment; filename="download.png"
...
TAG10 OK FETCH completed.
```

We can see in this code that some blocks are delimited by hex numbers between -- --, called parts. A message with parts is called a multipart message. It ' s possible to get such parts directly by passing the part index to the command.

In order to delete messages, the protocol has the commands STORE and EXPUNGE, which mark a message as deleted and commit such an operation.

```
TAG11 STORE 1 +FLAGS (\Deleted)
* 0 EXISTS
* 0 RECENT
TAG11 OK [READ-WRITE] SELECT completed; now in selected state
TAG12 EXPUNGE
EXPUNGE: TAG12 OK EXPUNGE completed
```

The last command is simple: NOOP. This command does nothing but is used to implement a keep-alive strategy. By default, the IMAP session closes after 30 minutes without commands. So, issuing the NOOP command keeps the connection active.

```
TAG17 NOOP
TAG17 OK NOOP completed.
```

And this finishes our IMAP overview. If you ' d like more information, there are a lot of good articles on the Web (I've selected some in resources section), and of course always remember the RFC 3501.

Resources

- Atmail ' s IMAP 101: Manual IMAP Sessions
- Fastmail ' s Why is IMAP better than POP?
- IETF ' s Internet Message Access Protocol
- IETF ' s Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
- Nylas ' Everything you need to know about IMAP

Next part, you'll use those commands within IRIS and see them in action!

See you!

#InterSystems IRIS