Network Working Group B. Leiba
Internet Draft IBM T.J. Watson Research Center
Document: draft-leiba-imap-implement-guide-00.txt September 1997

# IMAP4 Implementation Recommendations

## Status of this Document



## 1. Abstract

The IMAP4 specification [RFC-2060] describes a rich protocol for use in building clients and servers for storage, retrieval, and manipulation of electronic mail. Because the protocol is so rich and has so many implementation choices, there are often trade-offs that must be made and issues that must be considered when designing such clients and servers. This document attempts to outline these issues and to make recommendations in order to make the end products as interoperable as possible.

## 2. Conventions used in this document

In examples,"C:" indicates lines sent by a client that is connected to a server. "S:" indicates lines sent by the server to the client.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

## 3. Interoperability Issues and Recommendations

### 3.1. Accessibility

This section describes the issues related to access to servers and server resources. Concerns here include data sharing and maintenance of client/server connections.

#### 3.1.1. Multiple Accesses of the Same Mailbox

One strong point of IMAP4 is that, unlike POP3, it allows for multiple simultaneous access to a single mailbox. A user can, thus, read mail from a client at home while the client in the office is still connected; or the help desk staff can all work out of the same inbox, all seeing the same pool of questions. An important point about this capability, though is that NO SERVER IS GUARANTEED TO SUPPORT THIS. If you are selecting an IMAP server and this facility is important to you, be sure that the server you choose to install, in the configuration you choose to use, supports it.

If you are designing a client, you MUST NOT assume that you can access the same mailbox more than once at a time. That means
1. you must handle gracefully the failure of a SELECT command if the server refuses the second SELECT,
2. you must handle reasonably the severing of your connection (see "Severed Connections", below) if the server chooses to allow the second SELECT by forcing the first off,
3. you must avoid making multiple connections to the same mailbox in your own client (for load balancing or other such reasons), and
4. you must avoid using the STATUS command on a mailbox that you have selected (with some server implementations the STATUS command has the same problems with multiple access as do the SELECT and

EXAMINE commands).

A further note about STATUS: The STATUS command is sometimes used to check a non-selected mailbox for new mail. This mechanism MUST NOT be used to check for new mail in the selected mailbox; section 5.2 of [RFC-2060] specifically forbids this in its last paragraph.





### 3.1.2. Severed Connections

The client/server connection may be severed for one of three reasons: the client severs the connection, the server severs the connection, or the connection is severed by outside forces beyond the control of the client and the server (a telephone line drops, for example). Clients and servers must both deal with these situations.

When the client wants to sever a connection, it's usually because it has finished the work it needed to do on that connection. The client SHOULD send a LOGOUT command, wait for the tagged response, and then close the socket. But note that, while this is what's intended in the protocol design, there isn't universal agreement here. Some contend that sending the LOGOUT and waiting for the two responses (untagged BYE and tagged OK) is wasteful and unnecessary, and that the client can simply close the socket. The server should interpret the closed socket as a log out by the client. The counterargument is that it's useful from the standpoint of cleanup, problem determination, and the like, to have an explicit client log out.

Because of this disagreement, server designers must be aware that some clients might close the socket without sending a LOGOUT. In any case, whether or not a LOGOUT was sent, the server SHOULD NOT implicitly expunge any messages from the selected mailbox. If a client wants the server to do so, it MUST send a CLOSE or EXPUNGE command explicitly.

When the server wants to sever a connection it's usually due to an inactivity timeout or is because a situation has arisen that has changed the state of the mail store in a way that the server can not communicate to the client. The server SHOULD send an untagged BYE response to the client and then close the socket. Sending an untagged BYE response before severing allows the server to send a

human-readable explanation of the problem to the client, which the client may then log, display to the user, or both (see section 7.1.5 of [RFC-2060]). Note that the server MAY also choose to send an untagged ALERT as well, if it wants to be sure that the client shows the message to the user. The server designer should think carefully, though, before making a decision that's better left to the client.

## 3.2. Scaling

IMAP4 has many features that allow for scalability, as mail stores become larger and more numerous. Large numbers of users, mailboxes, and messages, and very large messages require thought to handle efficiently. This document will not address the administrative





issues involved in large numbers of users, but we will look at the other items.

### 3.2.1. Flood Control

There are three situations when a client can make a request that will result in a very large response - too large for the client reasonably to deal with: there are a great many mailboxes available, there are a great many messages in the selected mailbox, or there is a very large message part. The danger here is that the end user will be stuck waiting while the server sends (and the client processes) an enormous response. In all of these cases there are things a client can do to reduce that danger.

#### 3.2.1.1. Listing Mailboxes

Some servers present Usenet newsgroups to IMAP users. Newsgroups, and other such hierarchical mailbox structures, can be very numerous but may have only a few entries at the top level of hierarchy. Clients that will have trouble with this are those that use

```
C: 001 LIST "" *
```

to determine the mailbox list. Because of this, clients SHOULD NOT use an unqualified "*" that way in the LIST command. A safer approach is to list each level of hierarchy individually, allowing the user to traverse the tree one limb at a time, thus:

```
    C: 001 LIST "" %
    S: * LIST () "/" Banana
    S: * LIST ...etc...
    S: 001 OK done
and hen
    C: 002 LIST "" Banana/%
    S: * LIST () "/" Banana/Apple
    S: * LIST ...etc...
    S: 002 OK done
```

Using this technique the client's user interface can give the user full flexibility without choking on the voluminous reply to "LIST *". Of course, it is still possible that the reply to

```
    C: 005 LIST "" alt.fan.celebrity.%
```

may be thousands of entries long, and there is, unfortunately, nothing the client can do to protect itself from that. This has not yet been a notable problem.





### 3.2.1.2. Fetching the List of Messages

When a client selects a mailbox, it is given a count, in the untagged EXISTS response, of the messages in the mailbox. This number can be very large. In such a case it might be unwise to use

```
    C: 004 FETCH 1:* ALL
```

to populate the user's view of the mailbox. A good method to avoid problems with this is to batch the requests, thus:

```
    C: 004 FETCH 1:50 ALL
    S: * 1 FETCH ...etc...
    S: 004 OK done
    C: 005 FETCH 51:100 ALL
    S: * 51 FETCH ...etc...
    S: 005 OK done
    C: 006 FETCH 101:150 ALL
    ...etc...
```

Using this method, another command, such as "FETCH 6 BODY[1]" can be

inserted as necessary, and the client will not have its access to the server blocked by a storm of FETCH replies. (Such a method could be reversed to fetch the LAST 50 messages first, then the 50 prior to that, and so on.)

#### 3.2.1.3. Fetching a Large Body Part

The issue here is similar to the one for a list of messages. In the BODYSTRUCTURE response the client knows the size, in bytes, of the body part it plans to fetch. Suppose this is a 70 MB video clip. The client can use partial fetches to retrieve the body part in pieces, avoiding the problem of an uninterruptible 70 MB literal coming back from the server:

```
C: 022 FETCH 3 BODY[1]<0.20000>
S: * 3 FETCH (FLAGS(\Seen) BODY[1]<0> {20000}
S: ...data...)
S: 022 OK done
C: 023 FETCH 3 BODY[1]<20001.20000>
S: * 3 FETCH (BODY[1]<20001> {20000}
S: ...data...)
S: 023 OK done
C: 024 FETCH 3 BODY[1]<40001.20000>
...etc...
```

#### 3.2.1.4. BODYSTRUCTURE vs. Entire Messages

Because FETCH BODYSTRUCTURE is necessary in order to determine the





number of body parts, and, thus, whether a message has "attachments", clients often use FETCH FULL as their normal method of populating the user's view of a mailbox. The benefit is that the client can display a paperclip icon or some such indication along with the normal message summary. However, this comes at a significant cost with some server configurations. The parsing needed to generate the FETCH BODYSTRUCTURE response may be time-consuming compared with that needed for FETCH ENVELOPE. The client developer should consider this issue when deciding whether the ability to add a paperclip icon is worth the tradeoff in performance, especially with large mailboxes.

Some clients, rather than using FETCH BODYSTRUCTURE, use FETCH BODY[] (or the equivalent FETCH RFC822) to retrieve the entire message. They then do the MIME parsing in the client. This may give the client slightly more flexibility in some areas (access, for instance, to header fields that aren't returned in the BODYSTRUCTURE and ENVELOPE responses), but it can cause severe performance problems by forcing the transfer of all body parts when the user might only want to see some of them - a user logged on by modem and reading a small text message with a large ZIP file attached may prefer to read the text only and save the ZIP file for later. Therefore, a client SHOULD NOT normally retrieve entire messages and SHOULD retrieve message body parts selectively.

### 3.2.2. Subscriptions

The client isn't the only entity that can get flooded: the end user, too, may need some flood control. The IMAP4 protocol provides such control in the form of subscriptions. Most servers support the SUBSCRIBE, UNSUBSCRIBE, and LSUB commands, and many users choose to narrow down a large list of available mailboxes by subscribing to the ones that they usually want to see. Clients, with this in mind, SHOULD give the user a way to see only subscribed mailboxes. A client that never uses the LSUB command takes a significant usability feature away from the user. Of course, the client would not want to hide the LIST command completely; the user needs to be able to go both ways.

### 3.2.3. Searching

IMAP SEARCH commands can become particularly troublesome (that is, slow) on mailboxes containing a large number of messages. So let's put a few things in perspective in that regard.

The flag searches SHOULD be fast. The flag searches (ALL, [UN]SEEN, [UN]ANSWERED, [UN]DELETED, [UN]DRAFT, [UN]FLAGGED, NEW, OLD, RECENT) are known to be used by clients for the client's own use (for





instance, some clients use "SEARCH UNSEEN" to find unseen mail and "SEARCH DELETED" to warn the user before expunging messages).

Other searches, particularly the text searches (HEADER, TEXT, BODY) are initiated by the user, rather than by the client itself, and somewhat slower performance can be tolerated, since the user is aware that the search is being done (and is probably aware that it might be time-consuming).

The client MAY allow other commands to be sent to the server while a SEARCH is in progress, but at the time of this writing there is little or no server support for parallel processing of multiple commands in the same session (and see "Multiple Accesses of the Same Mailbox" above for a description of the dangers of trying to work around this by doing your SEARCH in another session).

Another word about text searches: some servers, built on database back-ends with indexed search capabilities, may return search results that do not match the IMAP spec's "case-insensitive substring" requirements. While these servers are in violation of the protocol, there is little harm in the violation as long as the search results are used only to response to a user's request. Still, developers of such servers should be aware that they ARE violating the protocol, should think carefully about that behaviour, and MUST be certain that their servers respond accurately to the flag searches for the reasons outlined above.

## 3.3. Miscellaneous Protocol Considerations

We describe here a number of important protocol-related issues, the misunderstanding of which has caused significant interoperability problems in IMAP4 implementations. One general item is that every implementer should be certain to take note of and to understand section 2.2.2 and the preamble to section 7 of the IMAP4rev1 spec [RFC-2060].

### 3.3.1. UIDs and UIDVALIDITY

Servers that support existing back-end mail stores often have no good place to save UIDs for messages. Often the existing mail store will not have the concept of UIDs in the sense that IMAP has: strictly increasing, never re-issued, 32-bit integers. Some servers solve this by storing the UIDs in a place that's accessible to end users, allowing for the possibility that the users will delete them. Others solve it by re-assigning UIDs every time a mailbox is selected.

The server SHOULD maintain UIDs permanently for all messages if it

can. If that's not possible, the server MUST change the UIDVALIDITY value for the mailbox whenever any of the UIDs may have become invalid. Clients MUST recognize that the UIDVALIDITY has changed and MUST respond to that condition by throwing away any information that they have saved about UIDs in that mailbox. There have been many problems in this area when clients have failed to do this; in the worst case it will result in loss of mail when a client deletes the wrong piece of mail by using a stale UID.

It seems to be a common myth that "the UIDVALIDITY and the UID, taken together, form a 64-bit identifier that uniquely identifies a message on a server". This is absolutely NOT TRUE. There is no assurance that the UIDVALIDITY values of two mailboxes be different, so the UIDVALIDITY in no way identifies a mailbox. The ONLY purpose of UIDVALIDITY is, as its name indicates, to give the client a way to check the validity of the UIDs it has cached.

Under extreme circumstances (and this is extreme, indeed), the server may have to invalidate UIDs while a mailbox is in use by a client - that is, the UIDs that the client knows about in its active mailbox are no longer valid. In that case, since there is no way to communicate this to the client (and since this could result in a loss of mail, should the client use the old UIDs to refer to the wrong messages), the server MUST force the client to re-select the mailbox, at which time it will obtain a new UIDVALIDITY value. To do this, the server closes this client session (see "Severed Connections" above) and the client then reconnects and gets back in synch.

### 3.3.2. FETCH Responses

When a client asks for certain information in a FETCH command, the server MAY return the requested information in any order, not necessarily in the order that it was requested. Further, the server MAY return the information in separate FETCH responses and MAY also return information that was not explicitly requested (to reflect to the client changes in the state of the subject message). Some examples:

```
C: 001 FETCH 1 UID FLAGS INTERNALDATE
S: * 5 FETCH (FLAGS (\Deleted))
S: * 1 FETCH (FLAGS (\Seen) INTERNALDATE "..." UID 345)
S: 001 OK done
```

(In this case, the responses are in a different order. Also, the server returned a flag update for message 5, which wasn't part of the client's request.)

```
    C: 002 FETCH 2 UID FLAGS INTERNALDATE
    S: * 2 FETCH (INTERNALDATE "...")
    S: * 2 FETCH (UID 399)
    S: * 2 FETCH (FLAGS ())
    S: 002 OK done
```

(In this case, the responses are in a different order and were returned in separate responses.)

```
    C: 003 FETCH 2 BODY[1]
    S: * 2 FETCH (FLAGS (\Seen) BODY[1] {14}
    S: Hello world!
    S: )
    S: 003 OK done
```

(In this case, the FLAGS response was added by the server, since fetching the body part caused the server to set the \Seen flag.)

Because of this characteristic a client MUST be ready to receive any FETCH response at any time and should use that information to update its local information about the message to which the FETCH response refers. A client MUST NOT assume that any FETCH responses will come in any particular order, or even that any will come at all. If after receiving the tagged response for a FETCH command the client finds that it did not get all of the information requested, the client SHOULD send a NOOP command to the server to ensure that the server has an opportunity to send any pending EXPUNGE responses to the client (see [RFC-2180]).

### 3.3.3. RFC822.SIZE

Some back-end mail stores keep the mail in a canonical form, rather than retaining the original MIME format of the messages. This means that the server must reassemble the message to produce a MIME stream when a client does a fetch such as RFC822 or BODY[], requesting the entire message. It also may mean that the server has no convenient way to know the RFC822.SIZE of the message. Often, such a server will actually have to build the MIME stream to compute the size, only to throw the stream away and report the size to the client.

When this is the case, some servers have chosen to estimate the size, rather than to compute it precisely. Such an estimate allows the client to display an approximate size to the user and to use the estimate in flood control considerations (q.v.), but requires that the client not use the size for things such as allocation of buffers, because those buffers might then be too small to hold the actual MIME stream. Instead, use the size that's returned in the literal when you fetch the data.

The protocol requires that the RFC822.SIZE value returned by the





server be EXACT. Estimating the size is a protocol violation, and server designers must be aware that, despite the performance savings they might realize in using an estimate, this practice will cause some clients to fail in various ways. If possible, the server SHOULD compute the RFC822.SIZE for a particular message once, and then save it for later retrieval. If that's not possible, the server MUST compute the value exactly every time. Incorrect estimates do cause severe interoperability problems with some clients.

### 3.3.4. Expunged Messages

If the server allows multiple connections to the same mailbox, it is often possible for messages to be expunged in one client unbeknownst to another client. Since the server is not allowed to tell the client about these expunged messages in response to a FETCH command, the server may have to deal with the issue of how to return information about an expunged message. There was extensive discussion about this issue, and the results of that discussion are summarized in [RFC-2180]. See that reference for a detailed explanation and for recommendations.

### 3.3.5. The Namespace Issue

Namespaces are a very muddy area in IMAP4 implementation right now (see [NAMESPACE] for a proposal to clear the water a bit). Until the issue is resolved, the important thing for client developers to understand is that some servers provide access through IMAP to more than just the user's personal mailboxes, and, in fact, the user's personal mailboxes may be "hidden" somewhere in the user's default

hierarchy. The client, therefore, SHOULD provide a setting wherein the user can specify a prefix to be used when accessing mailboxes. If the user's mailboxes are all in "~/mail/", for instance, then the user can put that string in the prefix. The client would then put the prefix in front of any name pattern in the LIST and LSUB commands:

```
    C: 001 LIST "" ~/mail/%
```

(See also "Reference Names in the LIST Command" below.)

### 3.3.6. Creating Special-Use Mailboxes

It may seem at first that this is part of the namespace issue; it is not, and is only indirectly related to it. A number of clients like to create special-use mailboxes with particular names. Most commonly, clients with a "trash folder" model of message deletion want to create a mailbox with the name "Trash" or "Deleted". Some clients want to create a "Drafts" mailbox, an "Outbox" mailbox, or a





"Sent Mail" mailbox. And so on. There are two major interoperability problems with this practice:

1. different clients may use different names for mailboxes with similar functions (such as "Trash" and "Deleted"), or may manage the same mailboxes in different ways, causing problems if a user switches between clients and
2. there is no guarantee that the server will allow the creation of the desired mailbox.

The client developer is, therefore, well advised to consider carefully the creation of any special-use mailboxes on the server, and, further, the client MUST NOT require such mailbox creation - that is, if you do decide to do this, you MUST handle gracefully the failure of the CREATE command and behave reasonably when your special-use mailboxes do not exist and can not be created.

### 3.3.7. Reference Names in the LIST Command

Many implementers of both clients and servers are confused by the "reference name" on the LIST command. The reference name is intended to be used in much the way a "cd" (change directory) command is used on Unix, PC DOS, Windows, and OS/2 systems. That is, the mailbox

name is interpreted in much the same way as a file of that name would be found if one had done a "cd" command into the directory specified by the reference name. For example, in Unix we have the following:

```
> cd /u/jones/junk
> vi banana        [file is "/u/jones/junk/banana"]
> vi stuff/banana  [file is "/u/jones/junk/stuff/banana"]
> vi /etc/hosts    [file is "/etc/hosts"]
```

The interoperability problems with this, in practice, are several. First, while some IMAP servers are built on Unix or PC file systems, many others are not, and the file system semantics do not make sense in those configurations. Second, while some IMAP servers expose the underlying file system to the clients, others allow access only to the user's personal mailboxes, or to some other limited set of files, making such file-system-like semantics less meaningful. Third, because the IMAP spec leaves the interpretation of the reference name as "implementation-dependent", the various server implementations handle it in vastly differing ways, and fourth, many implementers simply do not understand it and misuse it, do not use it, or ignore it as a result.

The following statement gets somewhat into the religious issues that we've tried to avoid scrupulously here; so be it: because of the confusion around the reference name, its use by a client is a dangerous thing, prone to result in interoperability problems. There





are servers that interpret it as originally intended; there are servers that ignore it completely; there are servers that simply prepend it to the mailbox name (with or without inserting a hierarchy delimiter in between). Because a client can't know which of these four behaviours to expect, the safest route is to leave it empty and put the full mailbox name pattern in the mailbox name argument.

There is in no way universal agreement about the use or non-use of the reference name. The last words here are, "Be aware."

## 3.4. A Word About Testing

Since the whole point of IMAP is interoperability, and since interoperability can not be tested in a vacuum, the final

recommendation of this treatise is, "Test against EVERYTHING." Test your client against every server you can get an account on. Test your server with every client you can get your hands on. Many clients make limited test versions available on the Web for the downloading. Many server owners will give serious client developers guest accounts for testing. Contact them and ask. NEVER assume that because your client works with one or two servers, or because your server does fine with one or two clients, you will interoperate well in general.

In particular, in addition to everything else, be sure to test against the reference implementations: the PINE client, the University of Washington server, and the Cyrus server.

See the following URLs on the web for more information here:
IMAP Products and Sources: http://www.imap.org/products.html
IMC MailConnect: http://www.imc.org/imc-mailconnect

## 4. Security Considerations

This document describes behavior of servers that use the IMAP4 protocol, and as such, has the same security considerations as described in [RFC-2060].

## 5. References

[RFC-2060], Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, University of Washington, December 1996.

[RFC-2119], Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.





[RFC-2180], Gahrns, M., "IMAP4 Multi-Accessed Mailbox Practice", RFC 2180, Microsoft, July 1997.

[NAMESPACE], Gahrns, M. & Newman, C., "IMAP4 Namespace", draft document <draft-gahrns-imap-namespace-01.txt>, June 1997.

## 6. Acknowledgments

To be completed...

This document is the result of discussions on the IMAP4 mailing list and is meant to reflect consensus of this group. In particular, Mark Crispin was an active participant in the discussions or made suggestions to this document.

## 7. Author's Address

Barry Leiba
IBM T.J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532

Phone: 1-914-784-7941
Email: leiba@watson.ibm.com