Network Working Group
Request for Comments: 2180
Category: Informational

M. Gahrns
Microsoft

July 1997

IMAP4 Multi-Accessed Mailbox Practice

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

1. Abstract

IMAP4[RFC-2060] is rich client/server protocol that allows a client to access and manipulate electronic mail messages on a server. Within the protocol framework, it is possible to have differing results for particular client/server interactions. If a protocol does not allow for this, it is often unduly restrictive.

For example, when multiple clients are accessing a mailbox and one attempts to delete the mailbox, an IMAP4 server may choose to implement a solution based upon server architectural constraints or individual preference.

With this flexibility comes greater client responsibility. It is not sufficient for a client to be written based upon the behavior of a particular IMAP server. Rather the client must be based upon the behavior allowed by the protocol.

By documenting common IMAP4 server practice for the case of simultaneous client access to a mailbox, we hope to ensure the widest amount of inter-operation between IMAP4 clients and servers.

The behavior described in this document reflects the practice of some existing servers or behavior that the consensus of the IMAP mailing list has deemed to be reasonable. The behavior described within this document is believed to be [RFC-2060] compliant. However, this document is not meant to define IMAP4 compliance, nor is it an exhaustive list of valid IMAP4 behavior. [RFC-2060] must always be consulted to determine IMAP4 compliance, especially for server behavior not described within this document.

2. Conventions used in this document

In examples,"C1:", "C2:" and "C3:" indicate lines sent by 3 different clients (client #1, client #2 and client #3) that are connected to a server. "S1:", "S2:" and "S3:" indicated lines sent by the server to client #1, client #2 and client #3 respectively.

A shared mailbox, is a mailbox that can be used by multiple users.

A multi-accessed mailbox, is a mailbox that has multiple clients simultaneously accessing it.

A client is said to have accessed a mailbox after a successful SELECT or EXAMINE command.

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. Deletion/Renaming of a multi-accessed mailbox

If an external agent or multiple clients are accessing a mailbox, care must be taken when handling the deletion or renaming of the mailbox. Following are some strategies an IMAP server may choose to use when dealing with this situation.

3.1. The server MAY fail the DELETE/RENAME command of a multi-accessed mailbox

In some cases, this behavior may not be practical. For example, if a large number of clients are accessing a shared mailbox, the window in which no clients have the mailbox accessed may be small or non-existent, effectively rendering the mailbox undeletable or unrenamable.

Example:

<Client #1 and Client #2 have mailbox FOO accessed. Client #1 tries to DELETE the mailbox and is refused>

```
C1: A001 DELETE FOO
S1: A001 NO Mailbox FOO is in use by another user.
```

3.2. The server MAY allow the DELETE command of a multi-accessed mailbox, but keep the information in the mailbox available for those clients that currently have access to the mailbox.

When all clients have finished accessing the mailbox, it is permanently removed. For clients that do not already have access to the mailbox, the 'ghosted' mailbox would not be available. For example, it would not be returned to these clients in a subsequent LIST or LSUB command and would not be a valid mailbox argument to any other IMAP command until the reference count of clients accessing the mailbox reached 0.

In some cases, this behavior may not be desirable. For example if someone created a mailbox with offensive or sensitive information, one might prefer to have the mailbox deleted and all access to the information contained within removed immediately, rather than continuing to allow access until the client closes the mailbox.

Furthermore, this behavior, may prevent 'recycling' of the same mailbox name until all clients have finished accessing the original mailbox.

Example:

<Client #1 and Client #2 have mailbox FOO selected. Client #1 DELETEs mailbox FOO>

```
C1: A001 DELETE FOO
S1: A001 OK Mailbox FOO is deleted.
```

<Client #2 is still able to operate on the deleted mailbox>

```
C2: B001 STORE 1 +FLAGS (\Seen)
S2: * 1 FETCH FLAGS (\Seen)
S2: B001 OK STORE completed
```

<Client #3 which did not have access to the mailbox prior to the deletion by client #1 does not have access to the mailbox>

```
C3: C001 STATUS FOO (MESSAGES)
S3: C001 NO Mailbox does not exist
```

<Nor is client #3 able to create a mailbox with the name FOO, while the reference count is non zero>

```
C3: C002 CREATE FOO
S3: C002 NO Mailbox FOO is still in use. Try again later.
```

<Client #2 closes its access to the mailbox, no other clients have access to the mailbox FOO and reference count becomes 0>

```
C2: B002 CLOSE
S2: B002 OK CLOSE Completed
```

<Now that the reference count on FOO has reached 0, the mailbox name can be recycled>

```
C3: C003 CREATE FOO
S3: C003 OK CREATE Completed
```

3.3. The server MAY allow the DELETE/RENAME of a multi-accessed mailbox, but disconnect all other clients who have the mailbox accessed by sending a untagged BYE response.

A server may often choose to disconnect clients in the DELETE case, but may choose to implement a "friendlier" method for the RENAME case.

Example:

<Client #1 and Client #2 have mailbox FOO accessed. Client #1 DELETEs the mailbox FOO>

```
C1: A002 DELETE FOO
S1: A002 OK DELETE completed.
```

<Server disconnects all other users of the mailbox>

```
S2: * BYE Mailbox FOO has been deleted.
```

3.4. The server MAY allow the RENAME of a multi-accessed mailbox by simply changing the name attribute on the mailbox.

Other clients that have access to the mailbox can continue issuing commands such as FETCH that do not reference the mailbox name. Clients would discover the renaming the next time they referred to the old mailbox name. Some servers MAY choose to include the [NEWNAME] response code in their tagged NO response to a command that contained the old mailbox name, as a hint to the client that the operation can succeed if the command is issued with the new mailbox name.

Example:

<Client #1 and Client #2 have mailbox FOO accessed. Client #1 RENAMEs the mailbox.>

```
C1: A001 RENAME FOO BAR
S1: A001 OK RENAME completed.
```

<Client #2 is still able to do operations that do not reference the mailbox name>

```
C2: B001 FETCH 2:4 (FLAGS)
S2: * 2 FETCH . . .
S2: * 3 FETCH . . .
S2: * 4 FETCH . . .
S2: B001 OK FETCH completed
```

<Client #2 is not able to do operations that reference the mailbox name>

```
C2: B002 APPEND FOO {300} C2: Date: Mon, 7 Feb 1994
21:52:25 0800 (PST) C2: . . .  S2: B002 NO [NEWNAME FOO
BAR] Mailbox has been renamed
```

4. Expunging of messages on a multi-accessed mailbox

If an external agent or multiple clients are accessing a mailbox, care must be taken when handling the EXPUNGE of messages. Other clients accessing the mailbox may be in the midst of issuing a command that depends upon message sequence numbers. Because an EXPUNGE response can not be sent while responding to a FETCH, STORE or SEARCH command, it is not possible to immediately notify the client of the EXPUNGE. This can result in ambiguity if the client issues a FETCH, STORE or SEARCH operation on a message that has been EXPUNGED.

4.1. Fetching of expunged messages

Following are some strategies an IMAP server may choose to use when dealing with a FETCH command on expunged messages.

Consider the following scenario:

- Client #1 and Client #2 have mailbox FOO selected.
- There are 7 messages in the mailbox.
- Messages 4:7 are marked for deletion.
- Client #1 issues an EXPUNGE, to expunge messages 4:7

4.1.1. The server MAY allow the EXPUNGE of a multi-accessed mailbox but keep the messages available to satisfy subsequent FETCH commands until it is able to send an EXPUNGE response to each client.

In some cases, the behavior of keeping "ghosted" messages may not be desirable. For example if a message contained offensive or sensitive information, one might prefer to instantaneously remove all access to the information, regardless of whether another client is in the midst of accessing it.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 is still able to access the expunged messages because the server has kept a 'ghosted' copy of the messages until it is able to notify client #2 of the EXPUNGE>

```
C2: B001 FETCH 4:7 RFC822
S2: * 4 FETCH RFC822 . . . (RFC822 info returned)
S2: * 5 FETCH RFC822 . . . (RFC822 info returned)
S2: * 6 FETCH RFC822 . . . (RFC822 info returned)
S2: * 7 FETCH RFC822 . . . (RFC822 info returned)
S2: B001 OK FETCH Completed
```

<Client #2 issues a command where it can get notified of the EXPUNGE>

```
C2: B002 NOOP
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 3 EXISTS
S2: B002 OK NOOP Complete
```

<Client #2 no longer has access to the expunged messages>

```
C2: B003 FETCH 4:7 RFC822
S2: B003 NO Messages 4:7 are no longer available.
```

4.1.2 The server MAY allow the EXPUNGE of a multi-accessed mailbox, and on subsequent FETCH commands return FETCH responses only for non-expunged messages and a tagged NO.

After receiving a tagged NO FETCH response, the client SHOULD issue a NOOP command so that it will be informed of any pending EXPUNGE responses. The client may then either reissue the failed FETCH command, or by examining the EXPUNGE response from the NOOP and the FETCH response from the FETCH, determine that the FETCH failed because of pending expunges.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 attempts to FETCH a mix of expunged and non-expunged messages. A FETCH response is returned only for then non-expunged messages along with a tagged NO>

```
C2: B001 FETCH 3:5 ENVELOPE
S2: * 3 FETCH ENVELOPE . . . (ENVELOPE info returned)
S2: B001 NO Some of the requested messages no longer exist
```

<Upon receiving a tagged NO FETCH response, Client #2 issues a NOOP to be informed of any pending EXPUNGE responses>

```
C2: B002 NOOP
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 3 EXISTS
S2: B002 OK NOOP Completed.
```

<By receiving a FETCH response for message 3, and an EXPUNGE response that indicates messages 4:7 have been expunged, the client does not need to re-issue the FETCH>

4.1.3 The server MAY allow the EXPUNGE of a multi-accessed mailbox, and on subsequent FETCH commands return the usual FETCH responses for non-expunged messages, "NIL FETCH Responses" for expunged messages, and a tagged OK response.

If all of the messages in the subsequent FETCH command have been expunged, the server SHOULD return only a tagged NO. In this case, the client SHOULD issue a NOOP command so that it will be informed of any pending EXPUNGE responses. The client may then either reissue the failed FETCH command, or by examining the EXPUNGE response from the NOOP, determine that the FETCH failed because of pending expunges.

"NIL FETCH responses" are a representation of empty data as appropriate for the FETCH argument specified.

Example:

```
* 1 FETCH (ENVELOPE (NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL))
* 1 FETCH (FLAGS ())
* 1 FETCH (INTERNALDATE "00-Jan-0000 00:00:00 +0000")
* 1 FETCH (RFC822 "")
* 1 FETCH (RFC822.HEADER "")
* 1 FETCH (RFC822.TEXT "")
* 1 FETCH (RFC822.SIZE 0)
* 1 FETCH (BODY ("TEXT" "PLAIN" NIL NIL NIL "7BIT" 0 0)
* 1 FETCH (BODYSTRUCTURE ("TEXT" "PLAIN" NIL NIL NIL "7BIT" 0 0)
* 1 FETCH (BODY[<section>] "")
* 1 FETCH (BODY[<section>]<partial> "")
```

In some cases, a client may not be able to distinguish between "NIL FETCH responses" received because a message was expunged and those received because the data actually was NIL. For example, a * 5 FETCH (FLAGS ()) response could be received if no flags were set on message 5, or because message 5 was expunged. In a case of potential ambiguity, the client SHOULD issue a command such as NOOP to force the sending of the EXPUNGE responses to resolve any ambiguity.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 attempts to access a mix of expunged and non-expunged messages. Normal data is returned for non-expunged message, "NIL FETCH responses" are returned for expunged messages>

```
C2: B002 FETCH 3:5 ENVELOPE
S2: * 3 FETCH ENVELOPE . . . (ENVELOPE info returned)
S2: * 4 FETCH ENVELOPE (NIL NIL NIL NIL NIL NIL NIL NIL
      NIL NIL)
S2: * 5 FETCH ENVELOPE (NIL NIL NIL NIL NIL NIL NIL NIL
      NIL NIL)
S2: B002 OK FETCH Completed
```

<Client #2 attempts to FETCH only expunged messages and receives a tagged NO response>

```
C2: B002 FETCH 4:7 ENVELOPE
S2: B002 NO Messages 4:7 have been expunged.
```

4.1.4 To avoid the situation altogether, the server MAY fail the EXPUNGE of a multi-accessed mailbox

In some cases, this behavior may not be practical. For example, if a large number of clients are accessing a shared mailbox, the window in which no clients have the mailbox accessed may be small or non-existent, effectively rendering the message unexpungeable.

4.2. Storing of expunged messages

Following are some strategies an IMAP server may choose to use when dealing with a STORE command on expunged messages.

4.2.1 If the ".SILENT" suffix is used, and the STORE completed successfully for all the non-expunged messages, the server SHOULD return a tagged OK.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 tries to silently STORE flags on expunged and non-expunged messages. The server sets the flags on the non-expunged messages and returns OK>

```
C2: B001 STORE 1:7 +FLAGS.SILENT (\SEEN)
S2: B001 OK
```

4.2.2. If the ".SILENT" suffix is not used, and only expunged messages are referenced, the server SHOULD return only a tagged NO.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 tries to STORE flags only on expunged messages>

```
C2: B001 STORE 5:7 +FLAGS (\SEEN)
S2: B001 NO  Messages have been expunged
```

4.2.3. If the ".SILENT" suffix is not used, and a mixture of expunged and non-expunged messages are referenced, the server MAY set the flags and return a FETCH response for the non-expunged messages along with a tagged NO.

After receiving a tagged NO STORE response, the client SHOULD issue a NOOP command so that it will be informed of any pending EXPUNGE responses. The client may then either reissue the failed STORE command, or by examining the EXPUNGE responses from the NOOP and FETCH responses from the STORE, determine that the STORE failed because of pending expunges.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 tries to STORE flags on a mixture of expunged and non-expunged messages>

```
C2: B001 STORE 1:7 +FLAGS (\SEEN)
S2: * FETCH 1 FLAGS (\SEEN)
S2: * FETCH 2 FLAGS (\SEEN)
S2: * FETCH 3 FLAGS (\SEEN)
S2: B001 NO Some of the messages no longer exist.

C2: B002 NOOP
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 3 EXISTS
S2: B002 OK NOOP Completed.
```

<By receiving FETCH responses for messages 1:3, and an EXPUNGE response that indicates messages 4:7 have been expunged, the client does not need to re-issue the STORE>

4.2.4. If the ".SILENT" suffix is not used, and a mixture of expunged and non-expunged messages are referenced, the server MAY return an untagged NO and not set any flags.

After receiving a tagged NO STORE response, the client SHOULD issue a NOOP command so that it will be informed of any pending EXPUNGE responses. The client would then re-issue the STORE command after updating its message list per any EXPUNGE response.

If a large number of clients are accessing a shared mailbox, the window in which there are no pending expunges may be small or non-existent, effectively disallowing a client from setting the flags on all messages at once.

Example: (Building upon the scenario outlined in 4.1.)

<Client #2 tries to STORE flags on a mixture of expunged and non-expunged messages>

```
C2: B001 STORE 1:7 +FLAGS (\SEEN)
S2: B001 NO  Some of the messages no longer exist.
```

<Client #2 issues a NOOP to be informed of the EXPUNGED messages>

```
C2: B002 NOOP
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 4 EXPUNGE
S2: * 3 EXISTS
S2: B002 OK NOOP Completed.
```

<Client #2 updates its message list and re-issues the STORE on only those messages that have not been expunged>

```
C2: B003 STORE 1:3 +FLAGS (\SEEN) S2: * FETCH 1 FLAGS
(\SEEN) S2: * FETCH 2 FLAGS (\SEEN) S2: * FETCH 3 FLAGS
(\SEEN) S2: B003 OK  STORE Completed
```

4.3. Searching of expunged messages

A server MAY simply not return a search response for messages that have been expunged and it has not been able to inform the client about. If a client was expecting a particular message to be returned in a search result, and it was not, the client SHOULD issue a NOOP command to see if the message was expunged by another client.

4.4 Copying of expunged messages

COPY is the only IMAP4 sequence number command that is safe to allow an EXPUNGE response on. This is because a client is not permitted to cascade several COPY commands together. A client is required to wait and confirm that the copy worked before issuing another one.

4.4.1 The server MAY disallow the COPY of messages in a multi-access mailbox that contains expunged messages.

Pending EXPUNGE response(s) MUST be returned to the COPY command.

Example:

```
C: A001 COPY 2,4,6,8 FRED
S: * 4 EXPUNGE
S: A001 NO COPY rejected, because some of the requested
   messages were expunged
```

Note: Non of the above messages are copied because if a COPY command is unsuccessful, the server MUST restore the destination mailbox to its state before the COPY attempt.

4.4.2 The server MAY allow the COPY of messages in a multi-access mailbox that contains expunged messages.

Pending EXPUNGE response(s) MUST be returned to the COPY command. Messages that are copied are messages corresponding to sequence numbers before any EXPUNGE response.

Example:

```
C: A001 COPY 2,4,6,8 FRED
S: * 3 EXPUNGE
S: A001 OK COPY completed
```

In the above example, the messages that are copied to FRED are messages 2,4,6,8 at the start of the COPY command. These are equivalent to messages 2,3,5,7 at the end of the COPY command. The EXPUNGE response can’t take place until after the messages from the COPY command are identified (because of the "no expunge while no commands in progress" rule).

Example:

```
C: A001 COPY 2,4,6,8 FRED
S: * 4 EXPUNGE
S: A001 OK COPY completed
```

In the above example, message 4 was copied before it was expunged, and MUST appear in the destination mailbox FRED.

5. 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].

In particular, some described server behavior does not allow for the immediate deletion of information when a mailbox is accessed by multiple clients. This may be a consideration when dealing with sensitive information where immediate deletion would be preferred.

6. 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.

7. Acknowledgments

This document is the result of discussions on the IMAP4 mailing list and is meant to reflect consensus of this group. In particular, Raymond Cheng, Mark Crispin, Jim Evans, Erik Forsberg, Steve Hole, Mark Keasling, Barry Leiba, Syd Logan, John Mani, Pat Moran, Larry Osterman, Chris Newman, Bart Schaefer, Vladimir Vulovic, and Jack De Winter were active participants in this discussion or made suggestions to this document.

8. Author's Address

Mike Gahrns
Microsoft
One Microsoft Way
Redmond, WA, 98072

Phone: (206) 936-9833
EMail: mikega@microsoft.com