Network Working Group
Request for Comments: 4551
Updates: 3501
Category: Standards Track

A. Melnikov
Isode Ltd.
S. Hole
ACI WorldWide/MessagingDirect

June 2006

IMAP Extension for Conditional STORE Operation or Quick Flag Changes Resynchronization

Status of This Memo

Copyright Notice

Abstract

Often, multiple IMAP (RFC 3501) clients need to coordinate changes to a common IMAP mailbox. Examples include different clients working on behalf of the same user, and multiple users accessing shared mailboxes. These clients need a mechanism to synchronize state changes for messages within the mailbox. They must be able to guarantee that only one client can change message state (e.g., message flags) at any time. An example of such an application is use of an IMAP mailbox as a message queue with multiple dequeueing clients.

The Conditional Store facility provides a protected update mechanism for message state information that can detect and resolve conflicts between multiple writing mail clients.

The Conditional Store facility also allows a client to quickly resynchronize mailbox flag changes.

This document defines an extension to IMAP (RFC 3501).

Table of Contents

1. Introduction and Overview

The Conditional STORE extension is present in any IMAP4 implementation that returns "CONDSTORE" as one of the supported capabilities in the CAPABILITY command response.

An IMAP server that supports this extension MUST associate a positive unsigned 64-bit value called a modification sequence (mod-sequence) with every IMAP message. This is an opaque value updated by the server whenever a metadata item is modified. The server MUST guarantee that each STORE command performed on the same mailbox (including simultaneous stores to different metadata items from different connections) will get a different mod-sequence value. Also, for any two successful STORE operations performed in the same session on the same mailbox, the mod-sequence of the second completed operation MUST be greater than the mod-sequence of the first completed. Note that the latter rule disallows the use of the system clock as a mod-sequence, because if system time changes (e.g., an NTP [NTP] client adjusting the time), the next generated value might be less than the previous one.

Mod-sequences allow a client that supports the CONDSTORE extension to determine if a message metadata has changed since some known moment. Whenever the state of a flag changes (i.e., the flag is added where previously it wasn't set, or the flag is removed and before it was set) the value of the modification sequence for the message MUST be updated. Adding the flag when it is already present or removing when it is not present SHOULD NOT change the mod-sequence.

When a message is appended to a mailbox (via the IMAP APPEND command, COPY to the mailbox, or using an external mechanism) the server generates a new modification sequence that is higher than the highest modification sequence of all messages in the mailbox and assigns it to the appended message.

The server MAY store separate (per-message) modification sequence values for different metadata items. If the server does so, per-message mod-sequence is the highest mod-sequence of all metadata items for the specified message.

The server that supports this extension is not required to be able to store mod-sequences for every available mailbox. Section 3.1.2 describes how the server may act if a particular mailbox doesn't support the persistent storage of mod-sequences.

This extension makes the following changes to the IMAP4 protocol:

a) adds UNCHANGEDSINCE STORE modifier.

b) adds the MODIFIED response code which should be used with an OK response to the STORE command. (It can also be used in a NO response.)

c) adds a new MODSEQ message data item for use with the FETCH command.

d) adds CHANGEDSINCE FETCH modifier.

e) adds a new MODSEQ search criterion.

f) extends the syntax of untagged SEARCH responses to include mod-sequence.

g) adds new OK untagged responses for the SELECT and EXAMINE commands.

h) defines an additional parameter to SELECT/EXAMINE commands.

i) adds the HIGHESTMODSEQ status data item to the STATUS command.

A client supporting CONDSTORE extension indicates its willingness to receive mod-sequence updates in all untagged FETCH responses by issuing:

- a SELECT or EXAMINE command with the CONDSTORE parameter,
- a STATUS (HIGHESTMODSEQ) command,
- a FETCH or SEARCH command that includes the MODSEQ message data item,
- a FETCH command with the CHANGEDSINCE modifier, or
- a STORE command with the UNCHANGEDSINCE modifier.

The server MUST include mod-sequence data in all subsequent untagged FETCH responses (until the connection is closed), whether they were caused by a regular STORE, a STORE with UNCHANGEDSINCE modifier, or an external agent.

This document uses the term "CONDSTORE-aware client" to refer to a client that announces its willingness to receive mod-sequence updates as described above. The term "CONDSTORE enabling command" will refer any of the commands listed above. A future extension to this document may extend the list of CONDSTORE enabling commands. A first CONDSTORE enabling command executed in the session MUST cause the

server to return HIGHESTMODSEQ (Section 3.1.1) unless the server has sent NOMODSEQ (Section 3.1.2) response code when the currently selected mailbox was selected.

The rest of this document describes the protocol changes more rigorously.

2. Conventions Used in This Document

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 [KEYWORDS].

In examples, lines beginning with "S:" are sent by the IMAP server, and lines beginning with "C:" are sent by the client. Line breaks may appear in example commands solely for editorial clarity; when present in the actual message, they are represented by "CRLF".

Formal syntax is defined using ABNF [ABNF].

The term "metadata" or "metadata item" is used throughout this document. It refers to any system or user-defined keyword. Future documents may extend "metadata" to include other dynamic message data.

Some IMAP mailboxes are private, accessible only to the owning user. Other mailboxes are not, either because the owner has set an Access Control List [ACL] that permits access by other users, or because it is a shared mailbox. Let’s call a metadata item "shared" for the mailbox if any changes to the metadata items are persistent and visible to all other users accessing the mailbox. Otherwise, the metadata item is called "private". Note that private metadata items are still visible to all sessions accessing the mailbox as the same user. Also note that different mailboxes may have different metadata items as shared.

See Section 1 for the definition of a "CONDSTORE-aware client" and a "CONDSTORE enabling command".

3. IMAP Protocol Changes

3.1. New OK Untagged Responses for SELECT and EXAMINE

This document adds two new response codes, HIGHESTMODSEQ and NOMODSEQ. One of those response codes MUST be returned in the OK untagged response for a successful SELECT/EXAMINE command.

When opening a mailbox, the server must check if the mailbox supports the persistent storage of mod-sequences. If the mailbox supports the persistent storage of mod-sequences and the mailbox open operation succeeds, the server MUST send the OK untagged response including HIGHESTMODSEQ response code. If the persistent storage for the mailbox is not supported, the server MUST send the OK untagged response including NOMODSEQ response code instead.

3.1.1. HIGHESTMODSEQ Response Code

This document adds a new response code that is returned in the OK untagged response for the SELECT and EXAMINE commands. A server supporting the persistent storage of mod-sequences for the mailbox MUST send the OK untagged response including HIGHESTMODSEQ response code with every successful SELECT or EXAMINE command:

```
OK [HIGHESTMODSEQ <mod-sequence-value>]
```

where <mod-sequence-value> is the highest mod-sequence value of all messages in the mailbox. When the server changes UIDVALIDITY for a mailbox, it doesn't have to keep the same HIGHESTMODSEQ for the mailbox.

A disconnected client can use the value of HIGHESTMODSEQ to check if it has to refetch metadata from the server. If the UIDVALIDITY value has changed for the selected mailbox, the client MUST delete the cached value of HIGHESTMODSEQ. If UIDVALIDITY for the mailbox is the same, and if the HIGHESTMODSEQ value stored in the client's cache is less than the value returned by the server, then some metadata items on the server have changed since the last synchronization, and the client needs to update its cache. The client MAY use SEARCH MODSEQ (Section 3.4) to find out exactly which metadata items have changed. Alternatively, the client MAY issue FETCH with the CHANGEDSINCE modifier (Section 3.3.1) in order to fetch data for all messages that have metadata items changed since some known modification sequence.

Example 1:

```
C: A142 SELECT INBOX
S: * 172 EXISTS
```

```
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * OK [UIDNEXT 4392] Predicted next UID
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S: * OK [HIGHESTMODSEQ 715194045007]
S: A142 OK [READ-WRITE] SELECT completed
```

3.1.2. NOMODSEQ Response Code

A server that doesn't support the persistent storage of mod-sequences for the mailbox MUST send the OK untagged response including NOMODSEQ response code with every successful SELECT or EXAMINE command. A server that returned NOMODSEQ response code for a mailbox, which subsequently receives one of the following commands while the mailbox is selected:

- a FETCH command with the CHANGEDSINCE modifier,
- a FETCH or SEARCH command that includes the MODSEQ message data item, or
- a STORE command with the UNCHANGEDSINCE modifier

MUST reject any such command with the tagged BAD response.

Example 2:

```
C: A142 SELECT INBOX
S: * 172 EXISTS
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * OK [UIDNEXT 4392] Predicted next UID
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S: * OK [NOMODSEQ] Sorry, this mailbox format doesn't support
     modsequences
S: A142 OK [READ-WRITE] SELECT completed
```

3.2. STORE and UID STORE Commands

This document defines the following STORE modifier (see Section 2.5 of [IMAPABNF]):

UNCHANGEDSINCE <mod-sequence>

For each message specified in the message set, the server performs the following. If the mod-sequence of any metadata item of the

message is equal or less than the specified UNCHANGEDSINCE value, then the requested operation (as described by the message data item) is performed. If the operation is successful, the server MUST update the mod-sequence attribute of the message. An untagged FETCH response MUST be sent, even if the .SILENT suffix is specified, and the response MUST include the MODSEQ message data item. This is required to update the client's cache with the correct mod-sequence values. See Section 3.3.2 for more details.

However, if the mod-sequence of any metadata item of the message is greater than the specified UNCHANGEDSINCE value, then the requested operation MUST NOT be performed. In this case, the mod-sequence attribute of the message is not updated, and the message number (or unique identifier in the case of the UID STORE command) is added to the list of messages that failed the UNCHANGESINCE test.

When the server finished performing the operation on all the messages in the message set, it checks for a non-empty list of messages that failed the UNCHANGESINCE test. If this list is non-empty, the server MUST return in the tagged response a MODIFIED response code. The MODIFIED response code includes the message set (for STORE) or set of UIDs (for UID STORE) of all messages that failed the UNCHANGESINCE test.

Example 3:

All messages pass the UNCHANGESINCE test.

```
C: a103 UID STORE 6,4,8 (UNCHANGEDSINCE 12121230045)
    +FLAGS.SILENT (\Deleted)
S: * 1 FETCH (UID 4 MODSEQ (12121231000))
S: * 2 FETCH (UID 6 MODSEQ (12121230852))
S: * 4 FETCH (UID 8 MODSEQ (12121130956))
S: a103 OK Conditional Store completed
```

Example 4:

```
C: a104 STORE * (UNCHANGEDSINCE 12121230045) +FLAGS.SILENT
   (\Deleted $Processed)
S: * 50 FETCH (MODSEQ (12111230047))
S: a104 OK Store (conditional) completed
```

Example 5:

```
C: c101 STORE 1 (UNCHANGEDSINCE 12121230045) -FLAGS.SILENT
   (\Deleted)
S: * OK [HIGHESTMODSEQ 12111230047]
```

```
S: * 50 FETCH (MODSEQ (12111230048))
S: c101 OK Store (conditional) completed
```

HIGHESTMODSEQ response code was sent by the server presumably because this was the first CONDSTORE enabling command.

Example 6:

In spite of the failure of the conditional STORE operation for message 7, the server continues to process the conditional STORE in order to find all messages that fail the test.

```
C: d105 STORE 7,5,9 (UNCHANGEDSINCE 320162338)
    +FLAGS.SILENT (\Deleted)
S: * 5 FETCH (MODSEQ (320162350))
S: d105 OK [MODIFIED 7,9] Conditional STORE failed
```

Example 7:

Same as above, but the server follows the SHOULD recommendation in Section 6.4.6 of [IMAP4].

```
C: d105 STORE 7,5,9 (UNCHANGEDSINCE 320162338)
    +FLAGS.SILENT (\Deleted)
S: * 7 FETCH (MODSEQ (320162342) FLAGS (\Seen \Deleted))
S: * 5 FETCH (MODSEQ (320162350))
S: * 9 FETCH (MODSEQ (320162349) FLAGS (\Answered))
S: d105 OK [MODIFIED 7,9] Conditional STORE failed
```

Use of UNCHANGEDSINCE with a modification sequence of 0 always fails if the metadata item exists. A system flag MUST always be considered existent, whether it was set or not.

Example 8:

```
C: a102 STORE 12 (UNCHANGEDSINCE 0)
    +FLAGS.SILENT ($MDNSent)
S: a102 OK [MODIFIED 12] Conditional STORE failed
```

The client has tested the presence of the $MDNSent user-defined keyword.

Note: A client trying to make an atomic change to the state of a particular metadata item (or a set of metadata items) should be prepared to deal with the case when the server returns the MODIFIED response code if the state of the metadata item being watched hasn't changed (but the state of some other metadata item has). This is necessary, because some servers don't store separate mod-sequences

for different metadata items. However, a server implementation SHOULD avoid generating spurious MODIFIED responses for +FLAGS/-FLAGS STORE operations, even when the server stores a single mod-sequence per message. Section 5 describes how this can be achieved.

Unless the server has included an unsolicited FETCH to update client's knowledge about messages that have failed the UNCHANGEDSINCE test, upon receipt of the MODIFIED response code, the client SHOULD try to figure out if the required metadata items have indeed changed by issuing FETCH or NOOP command. It is RECOMMENDED that the server avoids the need for the client to do that by sending an unsolicited FETCH response (Examples 9 and 10).

If the required metadata items haven't changed, the client SHOULD retry the command with the new mod-sequence. The client SHOULD allow for a configurable but reasonable number of retries (at least 2).

Example 9:

In the example below, the server returns the MODIFIED response code without sending information describing why the STORE UNCHANGEDSINCE operation has failed.

```
C: a106 STORE 100:150 (UNCHANGEDSINCE 212030000000)
    +FLAGS.SILENT ($Processed)
S: * 100 FETCH (MODSEQ (303181230852))
S: * 102 FETCH (MODSEQ (303181230852))
...
S: * 150 FETCH (MODSEQ (303181230852))
S: a106 OK [MODIFIED 101] Conditional STORE failed
```

The flag $Processed was set on the message 101...

```
C: a107 NOOP
S: * 101 FETCH (MODSEQ (303011130956) FLAGS ($Processed))
S: a107 OK
```

Or the flag hasn't changed, but another has (note that this server behaviour is discouraged. Server implementers should also see Section 5)...

```
C: b107 NOOP
S: * 101 FETCH (MODSEQ (303011130956) FLAGS (\Deleted \Answered))
S: b107 OK
```

...and the client retries the operation for the message 101 with the updated UNCHANGEDSINCE value

```
C: b108 STORE 101 (UNCHANGEDSINCE 303011130956)
    +FLAGS.SILENT ($Processed)
S: * 101 FETCH (MODSEQ (303181230852))
S: b108 OK Conditional Store completed
```

Example 10:

Same as above, but the server avoids the need for the client to poll for changes.

The flag $Processed was set on the message 101 by another client...

```
C: a106 STORE 100:150 (UNCHANGEDSINCE 212030000000)
    +FLAGS.SILENT ($Processed)
S: * 100 FETCH (MODSEQ (303181230852))
S: * 101 FETCH (MODSEQ (303011130956) FLAGS ($Processed))
S: * 102 FETCH (MODSEQ (303181230852))
...
S: * 150 FETCH (MODSEQ (303181230852))
S: a106 OK [MODIFIED 101] Conditional STORE failed
```

Or the flag hasn't changed, but another has (note that this server behaviour is discouraged. Server implementers should also see Section 5)...

```
C: a106 STORE 100:150 (UNCHANGEDSINCE 212030000000)
    +FLAGS.SILENT ($Processed)
S: * 100 FETCH (MODSEQ (303181230852))
S: * 101 FETCH (MODSEQ (303011130956) FLAGS (\Deleted \Answered))
S: * 102 FETCH (MODSEQ (303181230852))
...
S: * 150 FETCH (MODSEQ (303181230852))
S: a106 OK [MODIFIED 101] Conditional STORE failed
```

...and the client retries the operation for the message 101 with the updated UNCHANGEDSINCE value

```
C: b108 STORE 101 (UNCHANGEDSINCE 303011130956)
    +FLAGS.SILENT ($Processed)
S: * 101 FETCH (MODSEQ (303181230852))
S: b108 OK Conditional Store completed
```

Or the flag hasn't changed, but another has (nice server behaviour. Server implementers should also see Section 5)...

```
C: a106 STORE 100:150 (UNCHANGEDSINCE 212030000000)
    +FLAGS.SILENT ($Processed)
```

```
S: * 100 FETCH (MODSEQ (303181230852))
S: * 101 FETCH (MODSEQ (303011130956) FLAGS ($Processed \Deleted
     \Answered))
S: * 102 FETCH (MODSEQ (303181230852))
...
S: * 150 FETCH (MODSEQ (303181230852))
S: a106 OK Conditional STORE completed
```

Example 11:

The following example is based on the example from the Section 4.2.3 of [RFC-2180] and demonstrates that the MODIFIED response code may be also returned in the tagged NO response.

Client tries to conditionally STORE flags on a mixture of expunged and non-expunged messages; one message fails the UNCHANGEDSINCE test.

```
C: B001 STORE 1:7 (UNCHANGEDSINCE 320172338) +FLAGS (\SEEN)
S: * 1 FETCH (MODSEQ (320172342) FLAGS (\SEEN))
S: * 3 FETCH (MODSEQ (320172342) FLAGS (\SEEN))
S: B001 NO [MODIFIED 2] Some of the messages no longer exist.

C: B002 NOOP
S: * 4 EXPUNGE
S: * 4 EXPUNGE
S: * 4 EXPUNGE
S: * 4 EXPUNGE
S: * 2 FETCH (MODSEQ (320172340) FLAGS (\Deleted \Answered))
S: B002 OK NOOP Completed.
```

By receiving FETCH responses for messages 1 and 3, and EXPUNGE responses that indicate that messages 4 through 7 have been expunged, the client retries the operation only for the message 2. The updated UNCHANGEDSINCE value is used.

```
C: b003 STORE 2 (UNCHANGEDSINCE 320172340) +FLAGS (\Seen)
S: * 2 FETCH (MODSEQ (320180050))
S: b003 OK Conditional Store completed
```

Note: If a message is specified multiple times in the message set, and the server doesn't internally eliminate duplicates from the message set, it MUST NOT fail the conditional STORE operation for the second (or subsequent) occurrence of the message if the operation completed successfully for the first occurrence. For example, if the client specifies:

```
e105 STORE 7,3:9 (UNCHANGEDSINCE 12121230045)
 +FLAGS.SILENT (\Deleted)
```

the server must not fail the operation for message 7 as part of processing "3:9" if it succeeded when message 7 was processed the first time.

Once the client specified the UNCHANGEDSINCE modifier in a STORE command, the server MUST include the MODSEQ fetch response data items in all subsequent unsolicited FETCH responses.

This document also changes the behaviour of the server when it has performed a STORE or UID STORE command and the UNCHANGEDSINCE modifier is not specified. If the operation is successful for a message, the server MUST update the mod-sequence attribute of the message. The server is REQUIRED to include the mod-sequence value whenever it decides to send the unsolicited FETCH response to all CONDSTORE-aware clients that have opened the mailbox containing the message.

Server implementers should also see Section 3.8 for additional quality of implementation issues related to the STORE command.

3.3. FETCH and UID FETCH Commands

3.3.1. CHANGEDSINCE FETCH Modifier

This document defines the following FETCH modifier (see Section 2.4 of [IMAPABNF]):

CHANGEDSINCE <mod-sequence>

CHANGEDSINCE FETCH modifier allows to create a further subset of the list of messages described by sequence set. The information described by message data items is only returned for messages that have mod-sequence bigger than <mod-sequence>.

When CHANGEDSINCE FETCH modifier is specified, it implicitly adds MODSEQ FETCH message data item (Section 3.3.2).

Example 12:

```
C: s100 UID FETCH 1:* (FLAGS) (CHANGEDSINCE 12345)
S: * 1 FETCH (UID 4 MODSEQ (65402) FLAGS (\Seen))
S: * 2 FETCH (UID 6 MODSEQ (75403) FLAGS (\Deleted))
S: * 4 FETCH (UID 8 MODSEQ (29738) FLAGS ($NoJunk $AutoJunk
     $MDNSent))
S: s100 OK FETCH completed
```

3.3.2. MODSEQ Message Data Item in FETCH Command

This extension adds a MODSEQ message data item to the FETCH command. The MODSEQ message data item allows clients to retrieve mod-sequence values for a range of messages in the currently selected mailbox.

Once the client specified the MODSEQ message data item in a FETCH request, the server MUST include the MODSEQ fetch response data items in all subsequent unsolicited FETCH responses.

Syntax: MODSEQ

The MODSEQ message data item causes the server to return MODSEQ fetch response data items.

Syntax: MODSEQ (<permsg-modsequence>)

MODSEQ response data items contain per-message mod-sequences.

The MODSEQ response data item is returned if the client issued FETCH with MODSEQ message data item. It also allows the server to notify the client about mod-sequence changes caused by conditional STOREs (Section 3.2) and/or changes caused by external sources.

Example 13:

```
C: a FETCH 1:3 (MODSEQ)
S: * 1 FETCH (MODSEQ (624140003))
S: * 2 FETCH (MODSEQ (624140007))
S: * 3 FETCH (MODSEQ (624140005))
S: a OK Fetch complete
```

In this example, the client requests per-message mod-sequences for a set of messages.

When a flag for a message is modified in a different session, the server sends an unsolicited FETCH response containing the mod-sequence for the message.

Example 14:

(Session 1, authenticated as a user "alex"). The user adds a shared flag \Deleted:

```
C: A142 SELECT INBOX
...
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Answered \Deleted \Seen \*)] Limited
```

```
   ...

   C: A160 STORE 7 +FLAGS.SILENT (\Deleted)
   S: * 7 FETCH (MODSEQ (2121231000))
   S: A160 OK Store completed
```

(Session 2, also authenticated as the user "alex"). Any changes to flags are always reported to all sessions authenticated as the same user as in the session 1.

```
   C: C180 NOOP
   S: * 7 FETCH (FLAGS (\Deleted \Answered) MODSEQ (12121231000))
   S: C180 OK Noop completed
```

(Session 3, authenticated as a user "andrew"). As \Deleted is a shared flag, changes in session 1 are also reported in session 3:

```
   C: D210 NOOP
   S: * 7 FETCH (FLAGS (\Deleted \Answered) MODSEQ (12121231000))
   S: D210 OK Noop completed
```

The user modifies a private flag \Seen in session 1...

```
   C: A240 STORE 7 +FLAGS.SILENT (\Seen)
   S: * 7 FETCH (MODSEQ (12121231777))
   S: A240 OK Store completed
```

...which is only reported in session 2...

```
   C: C270 NOOP
   S: * 7 FETCH (FLAGS (\Deleted \Answered \Seen) MODSEQ
        (12121231777))
   S: C270 OK Noop completed
```

...but not in session 3.

```
   C: D300 NOOP
   S: D300 OK Noop completed
```

And finally, the user removes flags \Answered (shared) and \Seen (private) in session 1.

```
   C: A330 STORE 7 -FLAGS.SILENT (\Answered \Seen)
   S: * 7 FETCH (MODSEQ (12121245160))
   S: A330 OK Store completed
```

Both changes are reported in the session 2...

```
C: C360 NOOP
S: * 7 FETCH (FLAGS (\Deleted) MODSEQ (12121245160))
S: C360 OK Noop completed
```

...and only changes to shared flags are reported in session 3.

```
C: D390 NOOP
S: * 7 FETCH (FLAGS (\Deleted) MODSEQ (12121245160))
S: D390 OK Noop completed
```

Server implementers should also see Section 3.8 for additional quality of implementation issues related to the FETCH command.

3.4. MODSEQ Search Criterion in SEARCH

The MODSEQ criterion for the SEARCH command allows a client to search for the metadata items that were modified since a specified moment.

Syntax: MODSEQ [<entry-name> <entry-type-req>] <mod-sequence-valzer>

Messages that have modification values that are equal to or greater than <mod-sequence-valzer>. This allows a client, for example, to find out which messages contain metadata items that have changed since the last time it updated its disconnected cache. The client may also specify <entry-name> (name of metadata item) and <entry-type-req> (type of metadata item) before <mod-sequence-valzer>. <entry-type-req> can be one of "shared", "priv" (private), or "all". The latter means that the server should use the biggest value among "priv" and "shared" mod-sequences for the metadata item. If the server doesn't store internally separate mod-sequences for different metadata items, it MUST ignore <entry-name> and <entry-type-req>. Otherwise, the server should use them to narrow down the search.

For a flag <flagname>, the corresponding <entry-name> has a form "/flags/<flagname>" as defined in [IMAPABNF]. Note that the leading "\" character that denotes a system flag has to be escaped as per Section 4.3 of [IMAP4], as the <entry-name> uses syntax for quoted strings.

If client specifies a MODSEQ criterion in a SEARCH command and the server returns a non-empty SEARCH result, the server MUST also append (to the end of the untagged SEARCH response) the highest mod-sequence for all messages being returned. See also Section 3.5.

Example 15:

```
C: a SEARCH MODSEQ "/flags/\\draft" all 620162338
S: * SEARCH 2 5 6 7 11 12 18 19 20 23 (MODSEQ 917162500)
S: a OK Search complete
```

In the above example, the message numbers of any messages containing the string "IMAP4" in the "value" attribute of the "/comment" entry and having a mod-sequence equal to or greater than 620162338 for the "\Draft" flag are returned in the search results.

Example 16:

```
C: t SEARCH OR NOT MODSEQ 720162338 LARGER 50000
S: * SEARCH
S: t OK Search complete, nothing found
```

3.5. Modified SEARCH Untagged Response

Data: zero or more numbers
mod-sequence value (omitted if no match)

This document extends syntax of the untagged SEARCH response to include the highest mod-sequence for all messages being returned.

If a client specifies a MODSEQ criterion in a SEARCH (or UID SEARCH) command and the server returns a non-empty SEARCH result, the server MUST also append (to the end of the untagged SEARCH response) the highest mod-sequence for all messages being returned. See Section 3.4 for examples.

3.6. HIGHESTMODSEQ Status Data Items

This document defines a new status data item:

HIGHESTMODSEQ

The highest mod-sequence value of all messages in the mailbox. This is the same value that is returned by the server in the HIGHESTMODSEQ response code in an OK untagged response (see Section 3.1.1). If the server doesn't support the persistent storage of mod-sequences for the mailbox (see Section 3.1.2), the server MUST return 0 as the value of HIGHESTMODSEQ status data item.

Example 17:

```
C: A042 STATUS blurdybloop (UIDNEXT MESSAGES HIGHESTMODSEQ)
S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292
     HIGHESTMODSEQ 7011231777)
S: A042 OK STATUS completed
```

3.7. CONDSTORE Parameter to SELECT and EXAMINE

The CONDSTORE extension defines a single optional select parameter, "CONDSTORE", which tells the server that it MUST include the MODSEQ fetch response data items in all subsequent unsolicited FETCH responses.

The CONDSTORE parameter to SELECT/EXAMINE helps avoid a race condition that might arise when one or more metadata items are modified in another session after the server has sent the HIGHESTMODSEQ response code and before the client was able to issue a CONDSTORE enabling command.

Example 18:

```
C: A142 SELECT INBOX (CONDSTORE)
S: * 172 EXISTS
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * OK [UIDNEXT 4392] Predicted next UID
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S: * OK [HIGHESTMODSEQ 715194045007]
S: A142 OK [READ-WRITE] SELECT completed, CONDSTORE is now enabled
```

3.8. Additional Quality-of-Implementation Issues

Server implementations should follow the following rule, which applies to any successfully completed STORE/UID STORE (with and without UNCHANGEDSINCE modifier), as well as to a FETCH command that implicitly sets \Seen flag:

> Adding the flag when it is already present or removing when it is not present SHOULD NOT change the mod-sequence.

This will prevent spurious client synchronization requests.

However, note that client implementers MUST NOT rely on this server behavior. A client can't distinguish between the case when a server has violated the SHOULD mentioned above, and that when one or more clients set and unset (or unset and set) the flag in another session.

4. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) [ABNF] notation. Elements not defined here can be found in the formal syntax of the ABNF [ABNF], IMAP [IMAP4], and IMAP ABNF extensions [IMAPABNF] specifications.

Except as noted otherwise, all alphabetic characters are case-insensitive. The use of upper- or lowercase characters to define token strings is for editorial clarity only. Implementations MUST accept these strings in a case-insensitive fashion.

```
capability          =/ "CONDSTORE"

status-att          =/ "HIGHESTMODSEQ"
                       ;; extends non-terminal defined in RFC 3501.

status-att-val      =/ "HIGHESTMODSEQ" SP mod-sequence-valzer
                       ;; extends non-terminal defined in [IMAPABNF].
                       ;; Value 0 denotes that the mailbox doesn't
                       ;; support persistent mod-sequences
                       ;; as described in Section 3.1.2

store-modifier      =/ "UNCHANGEDSINCE" SP mod-sequence-valzer
                       ;; Only a single "UNCHANGEDSINCE" may be
                       ;; specified in a STORE operation

fetch-modifier      =/ chgsince-fetch-mod
                       ;; conforms to the generic "fetch-modifier"
                       ;; syntax defined in [IMAPABNF].

chgsince-fetch-mod  = "CHANGEDSINCE" SP mod-sequence-value
                       ;; CHANGEDSINCE FETCH modifier conforms to
                       ;; the fetch-modifier syntax

fetch-att           =/ fetch-mod-sequence
                       ;; modifies original IMAP4 fetch-att

fetch-mod-sequence  = "MODSEQ"

fetch-mod-resp      = "MODSEQ" SP "(" permsg-modsequence ")"

msg-att-dynamic     =/ fetch-mod-resp
```

```
   search-key          =/ search-modsequence
                          ;; modifies original IMAP4 search-key
                          ;;
                          ;; This change applies to all commands
                          ;; referencing this non-terminal, in
                          ;; particular SEARCH.

   search-modsequence  = "MODSEQ" [search-modseq-ext] SP
                         mod-sequence-valzer

   search-modseq-ext   = SP entry-name SP entry-type-req

   resp-text-code      =/ "HIGHESTMODSEQ" SP mod-sequence-value /
                          "NOMODSEQ" /
                          "MODIFIED" SP set

   entry-name          = entry-flag-name

   entry-flag-name     = DQUOTE "/flags/" attr-flag DQUOTE
                          ;; each system or user defined flag <flag>
                          ;; is mapped to "/flags/<flag>".
                          ;;
                          ;; <entry-flag-name> follows the escape rules
                          ;; used by "quoted" string as described in
                          ;; Section 4.3 of [IMAP4], e.g., for the flag
                          ;; \Seen the corresponding <entry-name> is
                          ;; "/flags/\\seen", and for the flag
                          ;; $MDNSent, the corresponding <entry-name>
                          ;; is "/flags/$mdnsent".

   entry-type-resp     = "priv" / "shared"
                          ;; metadata item type

   entry-type-req      = entry-type-resp / "all"
                          ;; perform SEARCH operation on private
                          ;; metadata item, shared metadata item or both

   permsg-modsequence  = mod-sequence-value
                          ;; per message mod-sequence

   mod-sequence-value  = 1*DIGIT
                          ;; Positive unsigned 64-bit integer
                          ;; (mod-sequence)
                          ;; (1 <= n < 18,446,744,073,709,551,615)

   mod-sequence-valzer = "0" / mod-sequence-value

   search-sort-mod-seq = "(" "MODSEQ" SP mod-sequence-value ")"
```

```
   select-param        =/ condstore-param
                          ;; conforms to the generic "select-param"
                          ;; non-terminal syntax defined in [IMAPABNF].

   condstore-param     = "CONDSTORE"

   mailbox-data        =/ "SEARCH" [1*(SP nz-number) SP
                          search-sort-mod-seq]

   attr-flag           = "\\Answered" / "\\Flagged" / "\\Deleted" /
                         "\\Seen" / "\\Draft" / attr-flag-keyword /
                         attr-flag-extension
                          ;; Does not include "\\Recent"

   attr-flag-extension = "\\" atom
                          ;; Future expansion.  Client implementations
                          ;; MUST accept flag-extension flags.  Server
                          ;; implementations MUST NOT generate
                          ;; flag-extension flags except as defined by
                          ;; future standard or standards-track
                          ;; revisions of [IMAP4].

   attr-flag-keyword   = atom
```

5. Server Implementation Considerations

This section describes how a server implementation that doesn't store separate per-metadata mod-sequences for different metadata items can avoid sending the MODIFIED response to any of the following conditional STORE operations:

```
   +FLAGS
   -FLAGS
   +FLAGS.SILENT
   -FLAGS.SILENT
```

Note that the optimization described in this section can't be performed in case of a conditional STORE FLAGS operation.

Let's use the following example. The client has issued

```
   C: a106 STORE 100:150 (UNCHANGEDSINCE 212030000000)
      +FLAGS.SILENT ($Processed)
```

When the server receives the command and parses it successfully, it iterates through the message set and tries to execute the conditional STORE command for each message.

Each server internally works as a client, i.e., it has to cache the current state of all IMAP flags as it is known to the client. In order to report flag changes to the client, the server compares the cached values with the values in its database for IMAP flags.

Imagine that another client has changed the state of a flag \Deleted on the message 101 and that the change updated the mod-sequence for the message. The server knows that the mod-sequence for the mailbox has changed; however, it also knows that:

a) the client is not interested in \Deleted flag, as it hasn't included it in +FLAGS.SILENT operation; and

b) the state of the flag $Processed hasn't changed (the server can determine this by comparing cached flag state with the state of the flag in the database).

Therefore, the server doesn't have to report MODIFIED to the client. Instead, the server may set $Processed flag, update the mod-sequence for the message 101 once again and send an untagged FETCH response with new mod-sequence and flags:

```
S: * 101 FETCH (MODSEQ (303011130956)
   FLAGS ($Processed \Deleted \Answered))
```

See also Section 3.8 for additional quality-of-implementation issues.

6. Security Considerations

It is believed that the Conditional STORE extension doesn't raise any new security concerns that are not already discussed in [IMAP4]. However, the availability of this extension may make it possible for IMAP4 to be used in critical applications it could not be used for previously, making correct IMAP server implementation and operation even more important.

7. IANA Considerations

IMAP4 capabilities are registered by publishing a standards track or IESG approved experimental RFC. The registry is currently located at:

http://www.iana.org/assignments/imap4-capabilities

This document defines the CONDSTORE IMAP capability. IANA has added it to the registry accordingly.

8. References

8.1. Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[ABNF] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[IMAP4] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[IMAPABNF] Melnikov, A. and C. Daboo, "Collected Extensions to IMAP4 ABNF", RFC 4466, April 2006.

8.2. Informative References

[ACAP] Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.

[ACL] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, December 2005.

[ANN] Daboo, C. and R. Gellens, "IMAP ANNOTATE Extension", Work in Progress, March 2006.

[NTP] Mills, D., "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992.

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

9. Acknowledgements

Some text was borrowed from "IMAP ANNOTATE Extension" [ANN] by Randall Gellens and Cyrus Daboo and from "ACAP -- Application Configuration Access Protocol" [ACAP] by Chris Newman and John Myers.

Many thanks to Randall Gellens for his thorough review of the document.

The authors also acknowledge the feedback provided by Cyrus Daboo, Larry Greenfield, Chris Newman, Harrie Hazewinkel, Arnt Gulbrandsen, Timo Sirainen, Mark Crispin, Ned Freed, Ken Murchison, and Dave Cridland.

Authors' Addresses

Alexey Melnikov
Isode Limited
5 Castle Business Village
36 Station Road
Hampton, Middlesex
TW12 2BX,
United Kingdom

EMail: Alexey.Melnikov@isode.com

Steve Hole
ACI WorldWide/MessagingDirect
#1807, 10088 102 Ave
Edmonton, AB
T5J 2Z1
Canada

EMail: Steve.Hole@messagingdirect.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).