Internet Draft: IMAP Extension for Conditional STORE
Document: draft-ietf-imapext-condstore-01.txt
Expires: October 2003

A. Melnikov
S. Hole
ACI WorldWide/MessagingDirect

April 2003

IMAP Extension for Conditional STORE operation

Status of this Memo

Copyright Notice

0.1. Open issues

1). Should conditional STORE be atomic accross message set (i.e. either all messages in message set weren't changed since and conditional STORE succeeds or operation fails for all messages)?
This can be difficult to implement for some servers.

Is a server allowed to reply NO to a conditional STORE operation that contains more than one message? Do we need a special response code for this (probably yes).

0.2. Change History

Changes from draft-ietf-imapext-condstore-00

1. Dropped "/message" prefix in entry names as per decision in San Francisco.
2. Fixed ABNF for SEARCH and SORT untagged responses.
3. Changed "private" to "priv" to be consistent with ANNOTATE.
4. MODIFIED response code is now returned in OK response, not NO.
5. Added NOMODSEQ response code.

Changes from draft-melnikov-imap-condstore-09:

1. Some text clarifications based on suggestions by Harrie Hazewinkel
2. Added paragraph about mailbox locking and DOS when conditional STORE operation is performed on a large mailbox.
3. Fixed syntax of <entry-name> to match the ANNOTATE extention.
4. Added sentence that a system flag MUST always be considered existent, when UNCHANGEDSINCE 0 is used. Is this a good idea?
5. Clarified client behavior upon receipt of MODIFIED response code.
6. Updated ABNF to clarify where 0 is allowed as mod-sequence and where it is not.
7. Got rid of MODSEQ response code and return this data in the untagged SEARCH/SORT responses.
8. Updated RFC number for the IMAP4rev1 document.

Changes from -08 to -09:

1. Added an extended example about reporting regular (non-conditional) flag changes to other sessions.
2. Simplified FETCH MODSEQ syntax by removing per-metadata requests and responses.

Changes from -07 to -08:

1. Added note saying the change to UIDVALIDITY also invalidates HIGHESTMODSEQ.
2. Fixed several bugs in ABNF for STATUS and STORE commands.

Changes from -06 to -07:

1. Added clarification that when a server does command reordering, the second completed operation gets the higher mod sequence.
2. Renamed annotation type specifier "both" to "all" as per suggestion from Minneapolis meeting.
3. Removed PERFLAGMODSEQ capability, as it doesn't buy anything: a client has to work with both types of servers (i.e. servers that support per message per flag modseqs and servers that support only per message modseqs) anyway.
4. Per flag mod-sequences are optional for a server to return. Updated syntax.
5. Allow MODSEQ response code only as a result of SEARCH/SORT as suggested by John Myers. MODSEQ response code is not allowed after FETCH or STORE.

Changes from -05 to -06:

1. Replaced "/message/flags/system" with "/message/flags" to match ANNOTATE draft.
2. Extended FETCH/SEARCH/SORT syntax to allow for specifying whether an operation should be performed on a shared or a private annotation (or both).

3. Corrected some examples.

Changes from -04 to -05:
1. Added support for SORT extension.
2. Multiple language/spelling fixes by Randall Gellens.

Changes from -03 to -04:
1. Added text saying that MODSEQ fetch data items cause server to include MODSEQ data response in all subsuquent unsolicited FETCH responses.
2. Added "authors address" section.

Changes from -02 to -03:
1. Changed MODTIME untagged response to MODTIME response code.
2. Added MODTIME response code to the tagged OK response for SEARCH. Updated examples accordingly.
3. Changed rule for sending untagged FETCH response as a result of STORE when .SILENT prefix is used. If .SILENT prefix is used, server doesn't have to send untagged FETCH response, because MODTIME response code already contains modtime.
4. Renamed MODTIME to MODSEQ to make sure there is no confusion between mod-sequence and ACAP modtime.
5. Minor ABNF changes.
6. Minor language corrections.

Changes from -01 to -02:
1. Added MODTIME data item to STATUS command.
2. Added OK untagged response to SELECT/EXAMINE.
3. Clarified that MODIFIED response code contains list of UIDs for conditional UID STORE and message set for STORE.
4. Added per-message modtime.
5. Added PERFLAGMODTIME capability.
6. Fixed several bugs in examples.
7. Added more comments to ABNF.

Changes from -00 to -01:
1. Refreshed the list of Open Issues.
2. Changed "attr-name" to "entry-name", because modtime applies to entry, not attribute.
3. Added MODTIME untagged response.
4. Cleaned up ABNF.
5. Added "Acknowledgments" section.
6. Fixed some spelling mistakes.

Table of Contents

1 Abstract X
2 Conventions Used in This Document X
3 Introduction and Overview X
4 IMAP Protocol Changes X

4.1 New OK untagged responses for SELECT and EXAMINE X
4.1.1 HIGHESTMODSEQ response code X
4.1.2 NOMODSEQ response code X
4.2 STORE and UID STORE Commands X
4.3 MODSEQ message data item in FETCH Command X
4.4 MODSEQ search criterion in SEARCH X
4.5 MODSEQ Sort Criterion X
4.6 Modified SEARCH and SORT untagged responses X
4.7 HIGHESTMODSEQ status data items X
5 Formal Syntax X
6 Security Considerations X
7 References X
7.1 Normative References X
7.2 Informative References X
8 Acknowledgments X
9 Author's Addresses X
10 Full Copyright Statement X

1. Abstract

Often, multiple IMAP clients need to coordinate changes to a common IMAP mailbox. Examples include different clients for 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 or annotations) 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.

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] as modified by [IMAP4].

The term "metadata" or "metadata item" is used throughout this document. It refers to any system or user defined keyword or an annotation [ANNOTATE].

Some IMAP mailboxes are private, accessible only to the owning user. Other mailboxes are not, either because the owner has set an ACL [ACL] which 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.

3. Introduction and Overview

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

Every IMAP message has an associated positive unsigned 64-bit value called a modification sequence (mod-sequence). This is an opaque value updated by the server whenever a metadata item is modified. The value is intended to be used only for comparisons within a server. However, 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., a 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 and before 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.

When an annotation is added, modified or removed the corresponding message mod-sequence MUST be updated.

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 extention is not required to be able to store mod-sequences for every available mailbox. Section 4.1.2 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) extends the syntax of the STORE command to allow STORE modifiers

b) adds the MODIFIED response code which should be used with an OK response to the STORE command

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

d) adds a new MODSEQ search criterion

e) extends syntax of untagged SEARCH and SORT responses to include mod-sequence.

f) adds a new OK untagged responses for the SELECT and EXAMINE commands

g) adds the HIGHESTMODSEQ status data item to the STATUS command

h) adds a new MODSEQ sort criterion

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

4. IMAP Protocol Changes

4.1. 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 and EXAMINE commands.

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 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.

4.1.1. 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 flags and/or annotations 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 as described in section 4.4 to find out exactly which metadata items have changed.

```
Example:    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 20010715194045007]
            S: A142 OK [READ-WRITE] SELECT completed
```

4.1.2 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.

```
Example:    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
```

4.2. STORE and UID STORE Commands

```
    Arguments:  message set
                OPTIONAL store modifiers
                message data item name
                value for message data item

    Responses:  untagged responses: FETCH

    Result:     OK - store completed
                NO - store error: can't store that data
                BAD - command unknown or arguments invalid
```

This document extends the syntax of the STORE and UID STORE commands (see section 6.4.6 of [IMAP4]) to include an optional STORE modifier. The document defines the following modifier:

UNCHANGEDSINCE

If the mod-sequence of any metadata item specified in the STORE operation for any message in the message set is greater than the specified unchangedsince value, then the command MUST NOT change any flags. Instead, the server replies with the OK tagged response that includes 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:

```
  C: a101 STORE 7,5,9 (UNCHANGEDSINCE 20000320162338)
      +FLAGS.SILENT (\Deleted)
  S: a101 OK [MODIFIED 7,9] Conditional STORE failed
```

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 which fail the test.

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:

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

Note: A client trying to atomically change the state of a particular flag (or a set of flags) should be prepared to deal with the case when the server returns MODIFIED response code if the state of the flag being watched hasn't changed (but the state of some other flag has). This is necessary, because some servers don't store separate mod-sequences for different flags or annotations. However, server implementations are discouraged from doing that, as it is possible not to return spurious errors even when storing a single mod-sequence per message.

Upon the receipt of MODIFIED response code the client SHOULD try to figure out if the required flags have indeed changed. If they haven't the client SHOULD retry the command.

Example:

```
  C: a106 STORE 100:150 (UNCHANGEDSINCE 200212030000000)
      +FLAGS.SILENT ($Processed)
  S: a106 OK [MODIFIED 101] Conditional STORE failed
```

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

```
  C: a107 NOOP
  S: * 101 FETCH (MODSEQ (200303011130956) FLAGS ($Processed))
  S: a107 OK
```

... so the client retries the operation for the rest of the messages

```
  C: a108 STORE 100,102:150 (UNCHANGEDSINCE 200303011130956)
      +FLAGS.SILENT ($Processed)
  S: * 100 FETCH (MODSEQ (200303181230852))
  S: * 102 FETCH (MODSEQ (200303181230852))
  ...
  S: a108 OK Conditional Store completed
```

Or the flag hasn't changed ...

```
  C: b107 NOOP
  S: * 101 FETCH (MODSEQ (200303011130956) FLAGS (\Deleted \Answered))
  S: b107 OK
```

```
 ... and the client retries the operation for all messages

  C: b108 STORE 100:150 (UNCHANGEDSINCE 200212030000000)
      +FLAGS.SILENT ($Processed)
  S: * 100 FETCH (MODSEQ (200303181230852))
  S: * 101 FETCH (MODSEQ (200303181230852))
  S: * 102 FETCH (MODSEQ (200303181230852))
  ...
  S: b108 OK Conditional Store completed
```

If the operation is successful the server MUST update the mod-sequence attribute for every message that was changed. Untagged FETCH responses MUST be sent (even if .SILENT is specified) and each response MUST include MODSEQ message data item if its mod-sequence has changed. This is required to update clients cache with the correct mod-sequence values. See section 4.3 for more details.

Example:

```
  C: a103 UID STORE 6,4,8 (UNCHANGEDSINCE 200012121230045)
      +FLAGS.SILENT (\Deleted)
  S: * 1 FETCH (UID 4 MODSEQ (200012121231000))
  S: * 2 FETCH (UID 6 MODSEQ (200012101230852))
  S: * 4 FETCH (UID 8 MODSEQ (200012121130956))
  S: a103 OK Conditional Store completed
```

Example:

```
  C: a104 STORE * (UNCHANGEDSINCE 200012121230045) +FLAGS.SILENT
     (\Deleted $Processed)
  S: * 50 FETCH (MODSEQ (200012111230045))
  S: a104 OK Store (conditional) 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:

```
   a105 STORE 7,3:9 (UNCHANGEDSINCE 200012121230045)
    +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.

4.3. 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 4.2) and/or changes caused by external sources.

Example:

```
C: a FETCH 1:3 (MODSEQ)
S: * 1 FETCH (MODSEQ (20000624140003))
S: * 2 FETCH (MODSEQ (20000624140007))
S: * 3 FETCH (MODSEQ (20000624140005))
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:

(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 (200012121231000))
            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 (200012121231000))
            S: C180 OK Noop completed

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

            C: D210 NOOP
            S: * 7 FETCH (FLAGS (\Deleted \Answered) MODSEQ (200012121231000))
            S: D210 OK Noop completed

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

            C: A240 STORE 7 +FLAGS.SILENT (\Seen)
            S: * 7 FETCH (MODSEQ (200012121231777))
            S: A240 OK Store completed

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

            C: C270 NOOP
            S: * 7 FETCH (FLAGS (\Deleted \Answered \Seen) MODSEQ
(200012121231777))
            S: C270 OK Noop completed

        ... but not in the session 3.

            C: D300 NOOP
            S: D300 OK Noop completed

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

            C: A330 STORE 7 -FLAGS.SILENT (\Answered \Seen)
            S: * 7 FETCH (MODSEQ (200012121245160))
            S: A330 OK Store completed

        Both changes are reported in the session 2 ...

            C: C360 NOOP
            S: * 7 FETCH (FLAGS (\Deleted) MODSEQ (200012121245160))
            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 (200012121245160))
S: D390 OK Noop completed
```

4.4. 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 which 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 flags and annotations, 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 [ANNOTATE]. 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 return a MODSEQ response code in the tagged OK response. The MODSEQ response code covers all messages returned in the untagged SEARCH results. See also section 4.6.

Example:

```
C: a SEARCH MODSEQ "/flags/draft" all 20010320162338
            ANNOTATION "/comment" "value" "IMAP4"
S: * SEARCH 2 5 6 7 11 12 18 19 20 23 (MODSEQ 20010917162500)
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 20010320162338 for the "\Draft" flag are returned in the search results.

Example:

```
     C: a SEARCH OR NOT MODSEQ 20010320162338 LARGER 50000
     S: * SEARCH
     S: a OK Search complete, nothing found
```

4.5. MODSEQ Sort Criterion

If a server implementing CONDSTORE also implements the SORT extension as defined by [SORT], it MUST also support sorting on per-message mod-sequence.

Syntax: MODSEQ

If client specifies a MODSEQ search (as per section 4.4) or sort criterion in the SORT command and the server returns a non-empty SORT result, the server MUST also return a MODSEQ response code in the tagged OK response which covers all messages returned in untagged SORT responses. See also section 4.6.

Example:

```
    C: A282 SORT (SUBJECT MODSEQ) UTF-8 SINCE 1-Feb-2001
    S: * SORT 2 81 83 84 82 882 (MODSEQ 117)
    S: A282 OK SORT completed
```

Example:

```
    C: A283 SORT (SUBJECT REVERSE DATE) UTF-8 MODSEQ 21
    S: * SORT 6 3 4 5 2 (MODSEQ 125)
    S: A283 OK SORT completed
```

Example:

```
    C: A284 SORT (MODSEQ) KOI8-R OR NOT MODSEQ 20010320162338
            SUBJECT "Privet"
    S: * SORT
    S: A284 OK Sort complete, nothing found
```

4.6. Modified SEARCH and SORT untagged responses

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

This document extends syntax of the untagged SEARCH and SORT responses to include 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.

If client specifies a MODSEQ search or sort criterion in a SORT

(or UID SORT) command and the server returns a non-empty SORT result, the server MUST also append (to the end of the untagged SORT response) the highest mod-sequence for all messages being returned.

4.7. HIGHESTMODSEQ status data items

This document defines a new status data item:

HIGHESTMODSEQ
The highest mod-sequence value all messages in the mailbox. This is the same value that is returned by the server in the HIGHESTMODSEQ response code in OK untagged response (see section 4.1.1).

```
Example:    C: A042 STATUS blurdybloop (UIDNEXT MESSAGES HIGHESTMODSEQ)
            S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292
                 HIGHESTMODSEQ 200201011231777)
            S: A042 OK STATUS completed
```

5. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [ABNF].

Non-terminals referenced but not defined below are as defined by [IMAP4].

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

```
capability          =/ "CONDSTORE"

status              = "STATUS" SP mailbox SP
                      "(" status-att-req *(SP status-att-req) ")"
                      ;; redefine STATUS command syntax defined in [IMAP4]

status-att-req      = status-att / "HIGHESTMODSEQ"

status-rsp-info     = status-att SP number /
                      "HIGHESTMODSEQ" SP mod-sequence-value

store               = "STORE" SP set store-modifiers SP store-att-flags

store-modifiers     = [ SP "(" store-modifier *(SP store-modifier) ")" ]

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

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

    fetch-mod-sequence  = "MODSEQ"

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

    search-key          =/ search-modsequence
                           ;; modifies original IMAP4 search-key

    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          = '"' "/flags/" attr-flag '"'
                          ;; each system or user defined flag <flag>
                          ;; is mapped to "/flags/<flag>".
                          ;;
                          ;; <entry-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 ")"

    sort-key            =/ "MODSEQ"

;;Borrowed from IMAP4rev1 and modified accordingly:
```

```
mailbox-data        =/ "STATUS" SP mailbox SP "("
                       [status-rsp-info *(SP status-rsp-info)] ")" /
                       "SEARCH" [1*(SP nz-number) SP search_sort_mod_seq] /
                       "SORT" [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 this specification.

attr-flag-keyword   = atom
```

6. Security Considerations

As a conditional STORE operation must be atomic for a message set, an implementation may choose to use some kind of message or even mailbox level locking for the duration of the conditional STORE operation. Such implementation may suffer from a Deny of Service Attack when conditional STORE is executed on a large mailbox.

Other IMAP4 security issues can be found in Security Considerations section of [IMAP4].

7. References

7.1. Normative References

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

[ABNF] Crocker, Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium, Demon Internet Ltd, November 1997.

[IMAP4] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 3501, University of Washington, March 2003.

[ANNOTATE] Gellens, R., Daboo, C., "IMAP ANNOTATE Extension", work in progress.
<http://www.ietf.org/internet-drafts/draft-ietf-imapext-annotate-xx.txt>

[SORT] Crispin, M., "Internet Message Access Protocol -- SORT Extension", work in progress.
<http://www.ietf.org/internet-drafts/draft-crispin-imapext-sort-xx.txt>

7.2. Informative References

[ACAP] Newman, Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, Innosoft, Netscape, November 1997.
<ftp://ftp.isi.edu/in-notes/rfc2244.txt>

[ACL] Myers, "IMAP4 ACL extension", RFC 2086, Carnegie Mellon, January 1997.
<ftp://ftp.isi.edu/in-notes/rfc2086.txt>

[NTP] Mills, D, "Network Time Protocol (Version 3) Specification, Implementation and Analysis", RFC 1305, March 1992.
<ftp://ftp.isi.edu/in-notes/rfc1305.txt>

8. Acknowledgments

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

Many thanks to Randall Gellens for his comments on how CONDSTORE should interact with ANNOTATE extension and for thorough review of the document.

Authors also acknowledge the feedback provided by Cyrus Daboo, Larry Greenfield, Chris Newman, Harrie Hazewinkel, Arnt Gulbrandsen Timo Sirainen and Mark Crispin.

9. Author's Addresses

Alexey Melnikov
mailto: mel@messagingdirect.com

ACI WorldWide/MessagingDirect
59 Clarendon Road, Watford, Hertfordshire,
WD17 1FQ, United Kingdom

Steve Hole
mailto: Steve.Hole@messagingdirect.com

ACI WorldWide/MessagingDirect
#900, 10117 Jasper Avenue,
Edmonton, Alberta, T5J 1W8, CANADA

10. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.