Network Working Group
Internet-Draft
Obsoletes: 4551, 5162 (if approved)
Updates: 4315, 3501, 2683 (if approved)
Intended status: Standards Track
Expires: July 11, 2014

A. Melnikov
Isode Ltd
D. Cridland
Arcode Inc

January 7, 2014

IMAP Extensions for Conditional STORE Operation or Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC)

draft-ietf-qresync-rfc5162bis-07.txt

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 efficiently synchronize state changes for messages within the mailbox.

The Conditional Store facility provides a protected update mechanism for message state information -- for example, the mechanism can be used to guarantee that only one client can change message state at any time -- and a mechanism for requesting only changes to message state.

This document additionally defines another IMAP extension, Quick Resynchronization, which builds on the Conditional Store extension to provide an IMAP client the ability to fully resynchronize a mailbox as part of the SELECT/EXAMINE command, without the need for additional server-side state or client round-trips.

This document obsoletes RFC 4551 and RFC 5162. It updates RFC 4315, RFC 3501 and RFC 2683.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 11, 2014.

Table of Contents

1. Introduction

Often, multiple IMAP [RFC3501] clients need to coordinate changes to a common IMAP mailbox. Examples include different clients working on behalf of the same user, and client representing multiple users accessing shared mailboxes. These clients need a mechanism to synchronize state changes for messages within the mailbox. The Conditional Store ("CONDSTORE") facility allows a client to quickly resynchronize mailbox flag changes.

The Conditional Store facility also provides a protected update mechanism for message state information that can detect and resolve conflicts between multiple writing mail clients. The mechanism can be used to guarantee that only one client can change message state at any given time. For example, this can be used by multiple clients which treat a mailbox as a message queue.

The Conditional Store facility is provided by associating a modification sequence (mod-sequence) with every IMAP message. This is updated whenever metadata (such as a message flag) is modified.

The CONDSTORE extension is described in more details in Section 3.1.

The CONDSTORE extension gives a disconnected client the ability to quickly resynchronize IMAP flag changes for previously seen messages. This can be done using the CHANGEDSINCE FETCH modifier once a mailbox is opened. In order for the client to discover which messages have been expunged, the client still has to issue a UID FETCH or a UID SEARCH command. The QRESYNC ("quick resync") IMAP extension is an extension to CONDSTORE that allows a reconnecting client to perform full resynchronization, including discovery of expunged messages, in a single round-trip. QRESYNC also introduces a new response, VANISHED, that allows for a more compact representation of a list of expunged messages.

QRESYNC can be useful for mobile clients that can experience frequent disconnects caused by environmental factors (battery life, signal strength, etc.). Such clients need a way to quickly reconnect to the IMAP server, while minimizing delay experienced by the user as well as the amount of traffic generated by resynchronization.

By extending the SELECT command to perform the additional resynchronization, this also allows clients to reduce concurrent connections to the IMAP server held purely for the sake of avoiding the resynchronization.

The QRESYNC extension is described in more details in Section 3.2.

2. Requirements Notation

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 [RFC2119].

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. If a single "C:" or "S:" label applies to multiple lines, then the line breaks between those lines are for editorial clarity only and are not part of the actual protocol exchange. The five characters [...] means that something has been elided.

Formal syntax is defined using ABNF [RFC5234].

The term "metadata" or "metadata item" is used throughout this document. It refers to any system or user-defined keyword. If the

server supports IMAP ANNOTATE-EXPERIMENT-1 extension [RFC5257], then metadata also includes message annotations. 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 [RFC4314] 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".

Understanding of the IMAP message sequence numbers and UIDs and the EXPUNGE response [RFC3501] is essential when reading this document.

3. IMAP Protocol Changes

3.1. CONDSTORE extension

An IMAP server that supports CONDSTORE MUST associate a positive unsigned 63-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 direct 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.

(*) Note: RFC 4551 defined mod-sequences as unsigned 64-bit values. In order to make implementations on various platforms (such as Java) easier, this version of the document redefines them as unsigned 63-bit values. This reflects consensus among WG members, including server implementers.

These rules allow a client to list all metadata changes since a well known point in time, as well as to perform conditional metadata

modifications based on an assumption that metadata state hasn't changed for a particular message.

In particular, 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 accessible to the currently logged in user for the specified message.

The server that supports CONDSTORE is not required to be able to store mod-sequences for every available mailbox. Section 3.1.2.2 describes how the server may act if a particular mailbox doesn't support the persistent storage of mod-sequences.

CONDSTORE 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,
- a STORE command with the UNCHANGEDSINCE modifier, or
- an ENABLE command containing "CONDSTORE" as one of the parameters. (This requirement only applies to servers that also implement the ENABLE extension [RFC5161].)

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 to 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 with a mailbox selected MUST cause the server to return HIGHESTMODSEQ (Section 3.1.2.1) for the mailbox (if any is selected), unless the server has sent NOMODSEQ (Section 3.1.2.2) response code when the currently selected mailbox was selected.

3.1.1. Advertising support for CONDSTORE

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

3.1.2. New OK Untagged Responses for SELECT and EXAMINE

This document adds two new response codes, HIGHESTMODSEQ and NOMODSEQ. One of these two response codes MUST be returned in an OK untagged response for any successful SELECT/EXAMINE command issued after a CONDSTORE enabling 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 an OK untagged response including HIGHESTMODSEQ response code. If the persistent storage for the mailbox is not supported, the server MUST send an OK untagged response including NOMODSEQ response code instead.

3.1.2.1. HIGHESTMODSEQ Response Code

This document adds a new response code that is returned in an OK untagged response for the SELECT and EXAMINE commands. Once a CONDSTORE enabling command is issued a server supporting the persistent storage of mod-sequences for the mailbox MUST send an 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.

Note that some existing CONDSTORE servers don't start tracking mod-sequences or don't report them until after a CONDSTORE enabling command is issued. Because of that, client wishing to receive HIGHESTMODSEQ/NOMODSEQ information must first send a CONDSTORE enabling command, for example by using SELECT/EXAMINE with CONDSTORE parameter (see Section 3.1.8).

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 3.1.5) to find out exactly which metadata items have changed. Alternatively, the client MAY issue FETCH with the

CHANGEDSINCE modifier (Section 3.1.4.1) in order to fetch data for all messages that have metadata items changed since some known modification sequence.

```
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
```

Example 1

3.1.2.2. NOMODSEQ Response Code

Once a CONDSTORE enabling command is issued a server that doesn't support the persistent storage of mod-sequences for the mailbox MUST send an OK untagged response including NOMODSEQ response code with every successful SELECT or EXAMINE command. Note that some existing CONDSTORE servers don't return NOMODSEQ until after a CONDSTORE enabling command is issued. Because of that, client wishing to receive HIGHESTMODSEQ/NOMODSEQ information must first send a CONDSTORE enabling command, for example by using SELECT/EXAMINE with CONDSTORE parameter (see Section 3.1.8).

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 a tagged BAD response.

```
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
```

Example 2

3.1.3. STORE and UID STORE Commands

This document defines the following STORE modifier (see Section 2.5 of [RFC4466]):

UNCHANGEDSINCE <mod-sequence> For each message specified in the message set, the server performs the following. If the mod-sequence of every metadata item of the message affected by the STORE/UID STORE is equal to 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 3.1.4.2 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 UNCHANGEDSINCE test.

When the server finishes performing the operation on all the messages in the message set, it checks for a non-empty list of messages that failed the UNCHANGEDSINCE 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 UNCHANGEDSINCE test.

All messages pass the UNCHANGEDSINCE 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 (12121230956))
S: a103 OK Conditional Store completed
```

Example 3

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

Example 4

```
C: c101 STORE 50 (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 5

The failure of the conditional STORE operation for any particular message or messages (7 in this example) does not stop the server from finding all messages that fail the UNCHANGEDSINCE test. All such messages are returned in the MODIFIED response code.

```
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 6

Same as above, but the server follows the SHOULD recommendation in Section 6.4.6 of [RFC3501].

```
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 7

```
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.

Example 8

Note: A client trying to make an atomic change to the state of a particular metadata item (or a set of metadata items) MUST 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 3.1.12 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).

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
```

Example 9

Or the flag hasn't changed, but another has (note that this server behaviour is discouraged. Server implementers should also see Section 3.1.12)...

```
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
```

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
```

Example 10

Or the flag hasn't changed, but another has (note that this server behaviour is discouraged. Server implementers should also see Section 3.1.12)...

```
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 3.1.12)...

```
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
```

The following example is based on the example from the Section 4.2.3 of [RFC2180] 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) FLAGS (\SEEN \Flagged))
S: b003 OK Conditional Store completed
```

Example 11

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.

As specified in Section 1, once the client specified the UNCHANGEDSINCE modifier in a STORE command, the server starts including 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 3.1.11 for additional quality of implementation issues related to the STORE command.

3.1.4. FETCH and UID FETCH Commands

3.1.4.1. CHANGEDSINCE FETCH Modifier

This document defines the following FETCH modifier (see Section 2.4 of [RFC4466]):

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 3.1.4.2).

```
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
```

Example 12

3.1.4.2. MODSEQ Message Data Item in FETCH Command

CONDSTORE 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.

As specified in Section 1, once the client has specified the MODSEQ message data item in a FETCH request, the server starts including 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 3.1.3) and/or changes caused by external sources.

```
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.

Example 13

Servers that only support the CONDSTORE extension (and not QRESYNC) SHOULD comply with requirements from Section 3.2.4.

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, as demonstrated in Example 14. Note that when the server also supports the QRESYNC extension (Section 3.2.3) and a "CONDSTORE enabling command" has been issued, all FETCH responses in the Example 14 must also include UID FETCH items as prescribed by Section 3.2.4.

(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
```

Example 14

Server implementers should also see Section 3.1.11 for additional quality of implementation issues related to the FETCH command.

3.1.5. MODSEQ Search Criterion in SEARCH

The MODSEQ criterion for the SEARCH (or UID 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 last means that the server MUST use the biggest value among "priv" and "shared" mod-sequences for the metadata item. If the server doesn't store 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 [RFC4466]. Note that the leading "\" character that denotes a system flag has to be escaped as per Section 4.3 of [RFC3501], as the <entry-name> uses syntax for quoted strings.

If 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 also Section 3.1.6. Note that other IMAP extensions such as ESEARCH [RFC4731] can override this requirement (see Section 3.1.10 for more details.)

```
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 having a mod-sequence equal to or greater than 620162338 for the "\Draft" flag are returned in the search results.

Example 15

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

Example 16

3.1.6. 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 3.1.5 for examples.

3.1.7. 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 3.1.2.1). If the server doesn't support the persistent storage of mod-sequences for the mailbox (see Section 3.1.2.2), the server MUST return 0 as the value of HIGHESTMODSEQ status data item.

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

Example 17

3.1.8. 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.

```
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
```

Example 18

3.1.9. Interaction with IMAP SORT and THREAD extensions

MODSEQ Search Criterion (see Section 3.1.5) causes modifications to SORT [RFC5256] responses similar to modifications to SEARCH responses defined in Section 3.1.6:

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

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

If a client specifies a MODSEQ 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. Note that other IMAP extensions such as ESORT [RFC5267] can override this requirement (see Section 3.1.10 for more details.)

THREAD commands which include a MODSEQ Search Criterion return THREAD responses as specified in [RFC5256], i.e. THREAD responses are unchanged by the CONDSTORE extension.

3.1.10. Interaction with IMAP ESORT and ESEARCH extensions

If a client specifies a MODSEQ criterion in an extended SEARCH (or extended UID SEARCH) [RFC4731] command and the server returns a non-empty SEARCH result, the server MUST return the ESEARCH response containing the MODSEQ result option as defined in Section 3.2 of [RFC4731].

```
C: a SEARCH RETURN (ALL) MODSEQ 1234
S: * ESEARCH (TAG "a") ALL 1:3,5 MODSEQ 1236
S: a OK Extended SEARCH completed
```

Example 19

If a client specifies a MODSEQ criterion in an extended SORT (or extended UID SORT) [RFC5267] command and the server returns a non-empty SORT result, the server MUST return the ESEARCH response containing the MODSEQ result option defined in Section 3.2 of [RFC4731].

```
C: a SORT RETURN (ALL) (DATE) UTF-8 MODSEQ 1234
S: * ESEARCH (TAG "a") ALL 5,3,2,1 MODSEQ 1236
S: a OK Extended SORT completed
```

Example 20

3.1.11. 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.

3.1.12. CONDSTORE 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 3.1.11 for additional quality-of-implementation issues.

3.2. QRESYNC extension

All protocol changes and requirements specified for the CONDSTORE extension are also a part of the QRESYNC extension.

The QRESYNC extension puts additional requirements on a server implementing the CONDSTORE extension. Each mailbox that supports persistent storage of mod-sequences, i.e., for which the server would send a HIGHESTMODSEQ untagged OK response code on a successful SELECT /EXAMINE, MUST increment the per-mailbox mod-sequence when one or more messages are expunged due to EXPUNGE, UID EXPUNGE, CLOSE or MOVE [RFC6851]; the server MUST associate the incremented mod-sequence with the UIDs of the expunged messages. Additionally, if the server also supports the IMAP METADATA extension [RFC5464], it MUST increment the per-mailbox mod-sequence when SETMETADATA successfully changes an annotation on the corresponding mailbox.

Server implementing QRESYNC MUST send untagged events to client in a way that client doesn't lose any changes in case of connectivity loss. In particular this means that if server sends MODSEQ FETCH data items while EXPUNGE (or VANISHED) replies with lower mod-sequences are being delayed, the server MUST send HIGHESTMODSEQ response code with a lower value than the EXPUNGE's mod-sequence. See example in Section 6.

3.2.1. Impact to CONDSTORE-only clients

A client that supports CONDSTORE but not QRESYNC might resynchronize a mailbox and discover that its HIGHESTMODSEQ has increased from the value cached by the client. If the increase is only due to messages having been expunged since the client last synchronized, the client is likely to send a FETCH ... CHANGEDSINCE command that returns no data. Thus, a client that supports CONDSTORE but not QRESYNC might incur a penalty of an unneeded round-trip when resynchronizing some mailboxes (those that have had messages expunged but no flag changes since the last synchronization).

This extra round-trip is only incurred by clients that support CONDSTORE but not QRESYNC, and only when a mailbox has had messages expunged but no flag changes to non-expunged messages. Since CONDSTORE is a relatively new extension, it is thought likely that clients that support it will also support QRESYNC.

3.2.2. Advertising support for QRESYNC

The quick resync IMAP extension is present if an IMAP4 server returns "QRESYNC" as one of the supported capabilities to the CAPABILITY command.

For compatibility with clients that only support the CONDSTORE IMAP extension, servers SHOULD also advertise "CONDSTORE" in the CAPABILITY response.

3.2.3. Use of ENABLE

Servers supporting QRESYNC MUST implement and advertise support for the ENABLE [RFC5161] IMAP extension. Also, the presence of the "QRESYNC" capability implies support for the CONDSTORE IMAP extension even if the "CONDSTORE" capability isn't advertised. A server compliant with this specification is REQUIRED to support "ENABLE QRESYNC" and "ENABLE QRESYNC CONDSTORE" (which are "CONDSTORE enabling commands", see Section 1), and have identical results. Note that the order of parameters is not significant), but there is no requirement for a compliant server to support "ENABLE CONDSTORE" by itself. The "ENABLE QRESYNC"/"ENABLE QRESYNC CONDSTORE" command also tells the server that it MUST start sending VANISHED responses (see Section 3.2.10) instead of EXPUNGE responses for all mailboxes for which the server doesn't return the NOMODSEQ response code. This change remains in effect until the connection is closed.

A client making use of QRESYNC MUST issue "ENABLE QRESYNC" once it is authenticated. A server MUST respond with a tagged BAD response if the QRESYNC parameter to the SELECT/EXAMINE command or the VANISHED UID FETCH modifier is specified and the client hasn't issued "ENABLE QRESYNC", or the server has not positively responded (in the current connection) to that command with the untagged ENABLED response containing QRESYNC.

3.2.4. Additional requirements on QRESYNC servers

Once a "CONDSTORE enabling command" is issued by the client, the server MUST automatically include both UID and mod-sequence data in all subsequent untagged FETCH responses (until the connection is closed), whether they were caused by a regular STORE/UID STORE, a STORE/UID STORE with UNCHANGEDSINCE modifier, FETCH/UID FETCH that

implicitly set \Seen flag or an external agent. Note that this rule doesn't affect untagged FETCH responses caused by a FETCH command that doesn't include UID and/or MODSEQ FETCH data item (and doesn't implicitly set \Seen flag), or UID FETCH without the MODSEQ FETCH data item.

3.2.5. QRESYNC Parameter to SELECT/EXAMINE

The Quick Resynchronization parameter to SELECT/EXAMINE commands has four arguments:

- the last known UIDVALIDITY,
- the last known modification sequence,
- the optional set of known UIDs, and
- an optional parenthesized list of known sequence ranges and their corresponding UIDs.

A server MUST respond with a tagged BAD response if the Quick Resynchronization parameter to SELECT/EXAMINE command is specified and the client hasn't issued "ENABLE QRESYNC" in the current connection, or the server has not positively responded to that command with the untagged ENABLED response containing QRESYNC.

Before opening the specified mailbox, the server verifies all arguments for syntactic validity. If any parameter is not syntactically valid, the server returns the tagged BAD response, and the mailbox remains unselected. Once the check is done, the server opens the mailbox as if no SELECT/EXAMINE parameters are specified (this is subject to processing of other parameters as defined in other extensions). In particular this means that the server MUST send all untagged responses as specified in Sections 6.3.1 and 6.3.2 of [RFC3501].

After that, the server checks the UIDVALIDITY value provided by the client. If the provided UIDVALIDITY doesn't match the UIDVALIDITY for the mailbox being opened, then the server MUST ignore the remaining parameters and behave as if no dynamic message data changed. The client can discover this situation by comparing the UIDVALIDITY value returned by the server. This behavior allows the client not to synchronize the mailbox or decide on the best synchronization strategy.

Example: Attempting to resynchronize INBOX, but the provided UIDVALIDITY parameter doesn't match the current UIDVALIDITY value.

```
C: A02 SELECT INBOX (QRESYNC (67890007 20050715194045000
41,43:211,214:541))
S: * 464 EXISTS
S: * 3 RECENT
S: * OK [UIDVALIDITY 3857529045] UIDVALIDITY
S: * OK [UIDNEXT 550] Predicted next UID
S: * OK [HIGHESTMODSEQ 90060128194045007] Highest mailbox
mod-sequence
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft
\Deleted \Seen \*)] Permanent flags
S: A02 OK [READ-WRITE] Sorry, UIDVALIDITY mismatch
```

Modification Sequence and UID Parameters:

A server that doesn't support the persistent storage of mod-sequences for the mailbox MUST send an OK untagged response including the NOMODSEQ response code with every successful SELECT or EXAMINE command (see Section 3.1.2.2). Such a server doesn't need to remember mod-sequences for expunged messages in the mailbox. It MUST ignore the remaining parameters and behave as if no dynamic message data changed.

If the provided UIDVALIDITY matches that of the selected mailbox, the server then checks the last known modification sequence.
The server sends the client any pending flag changes (using FETCH responses that MUST contain UIDs) and expunges those that have occurred in this mailbox since the provided modification sequence.

If the list of known UIDs was also provided, the server should only report flag changes and expunges for the specified messages. If the client did not provide the list of UIDs, the server acts as if the client has specified "1:<maxuid>", where <maxuid> is the mailbox's UIDNEXT value minus 1. If the mailbox is empty and never had any messages in it, then lack of the list of UIDs is interpreted as an empty set of UIDs.

Thus, the client can process just these pending events and need not perform a full resynchronization. Without the message sequence number matching information, the result of this step is semantically equivalent to the client issuing:
tag1 UID FETCH "known-uids" (FLAGS) (CHANGEDSINCE "mod-sequence-value" VANISHED)

In particular this means that all requirement specified in Section 3.2.6 apply.

Example:

Example:

```
C: A03 SELECT INBOX (QRESYNC (67890007
   90060115194045000 41:211,214:541))
S: * OK [CLOSED]
S: * 100 EXISTS
S: * 11 RECENT
S: * OK [UIDVALIDITY 67890007] UIDVALIDITY
S: * OK [UIDNEXT 600] Predicted next UID
S: * OK [HIGHESTMODSEQ 90060115205545359] Highest
   mailbox mod-sequence
S: * OK [UNSEEN 7] There are some unseen
   messages in the mailbox
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft
   \Deleted \Seen \*)] Permanent flags
S: * VANISHED (EARLIER) 41,43:116,118,120:211,214:540
S: * 49 FETCH (UID 117 FLAGS (\Seen \Answered) MODSEQ
   (90060115194045001))
S: * 50 FETCH (UID 119 FLAGS (\Draft $MDNSent) MODSEQ
   (90060115194045308))
S: * 51 FETCH (UID 541 FLAGS (\Seen $Forwarded) MODSEQ
   (90060115194045001))
S: A03 OK [READ-WRITE] mailbox selected
```

In the above example flag information for the UID 42 is not returned, presumably because its flags haven't changed since the MODSEQ 90060115194045000.

Message sequence match data:

A client MAY provide a parenthesized list of a message sequence set and the corresponding UID sets. Both MUST be provided in ascending order. The server uses this data to restrict the range for which it provides expunged message information.

Conceptually, the client provides a small sample of sequence numbers for which it knows the corresponding UIDs. The server then compares each sequence number and UID pair the client provides with the current state of the mailbox. If a pair matches, then the client knows of any expunges up to, and including, the message, and thus will not include that range in the VANISHED response, even if the "mod-sequence-value" provided by the client is too old for the server to have data of when those messages were expunged.

Thus, if the Nth message number in the first set in the list is 4, and the Nth UID in the second set in the list is 8, and the mailbox's fourth message has UID 8, then no UIDs equal to or less than 8 are present in the VANISHED response. If the (N+1)th message number is 12, and the (N+1)th UID is 24, and the (N+1)th message in the mailbox has UID 25, then the lowest UID included in the VANISHED response would be 9.

In the following two examples, the server is unable to remember expunges at all, and only UIDs with messages divisible by three are present in the mailbox. In the first example, the client does not use the fourth parameter; in the second, it provides it. This example is somewhat extreme, but shows that judicious usage of the sequence match data can save a substantial amount of bandwidth.

Example:

Example:

```
C: A04 SELECT INBOX (QRESYNC (67890007
   90060115194045000 1:29997))
S: * 10003 EXISTS
S: * 4 RECENT
S: * OK [UIDVALIDITY 67890007] UIDVALIDITY
S: * OK [UIDNEXT 30013] Predicted next UID
S: * OK [HIGHESTMODSEQ 90060115205545359] Highest mailbox mod-
   sequence
S: * OK [UNSEEN 7] There are some unseen messages in the mailbox
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft
   \Deleted \Seen \*)] Permanent flags
S: * VANISHED (EARLIER) 1:2,4:5,7:8,10:11,13:14,[...],
   29668:29669,29671:29996
S: * 1 FETCH (UID 3 FLAGS (\Seen \Answered $Important) MODSEQ
   (90060115194045001))
S: ...
S: * 9889 FETCH (UID 29667 FLAGS (\Seen \Answered) MODSEQ
   (90060115194045027))
S: * 9890 FETCH (UID 29670 FLAGS (\Draft $MDNSent) MODSEQ
   (90060115194045028))
S: ...
S: * 9999 FETCH (UID 29997 FLAGS (\Seen $Forwarded) MODSEQ
   (90060115194045031))
S: A04 OK [READ-WRITE] mailbox selected
```

Example:

Example:

```
C: B04 SELECT INBOX (QRESYNC (67890007
   90060115194045000 1:29997 (5000,7500,9000,9990:9999 15000,
   22500,27000,29970,29973,29976,29979,29982,29985,29988,29991,
   29994,29997)))
S: * 10003 EXISTS
S: * 4 RECENT
S: * OK [UIDVALIDITY 67890007] UIDVALIDITY
S: * OK [UIDNEXT 30013] Predicted next UID
S: * OK [HIGHESTMODSEQ 90060115205545359] Highest mailbox mod-
   sequence
S: * OK [UNSEEN 7] There are some unseen messages in the mailbox
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft
   \Deleted \Seen \*)] Permanent flags
S: * 1 FETCH (UID 3 FLAGS (\Seen \Answered $Important) MODSEQ
   (90060115194045001))
S: ...
S: * 9889 FETCH (UID 29667 FLAGS (\Seen \Answered) MODSEQ
   (90060115194045027))
S: * 9890 FETCH (UID 29670 FLAGS (\Draft $MDNSent) MODSEQ
   (90060115194045028))
S: ...
S: * 9999 FETCH (UID 29997 FLAGS (\Seen $Forwarded) MODSEQ
   (90060115194045031))
S: B04 OK [READ-WRITE] mailbox selected
```

3.2.6. VANISHED UID FETCH Modifier

[RFC4466] has extended the syntax of the FETCH and UID FETCH commands to include an optional FETCH modifier. This document defines a new UID FETCH modifier: VANISHED.

Note, that the VANISHED UID FETCH modifier is NOT allowed with a FETCH command. The server MUST return a tagged BAD response if this response is specified as a modifier to the FETCH command.

A server MUST respond with a tagged BAD response if the VANISHED UID FETCH modifier is specified and the client hasn't issued "ENABLE QRESYNC" in the current connection.

The VANISHED UID FETCH modifier MUST only be specified together with the CHANGEDSINCE UID FETCH modifier. If the VANISHED UID FETCH modifier is used without the CHANGEDSINCE UID FETCH modifier the server MUST respond with a tagged BAD response.

The VANISHED UID FETCH modifier instructs the server to report those messages from the UID set parameter that have been expunged and whose associated mod-sequence is larger than the specified mod-sequence.

That is, the client requests to be informed of messages from the specified set that were expunged since the specified mod-sequence. Note that the mod-sequence(s) associated with these messages were updated when the messages were expunged (as described above). The expunged messages are reported using the VANISHED response as described in Section 3.2.10, which MUST contain the EARLIER tag. Any VANISHED (EARLIER) responses MUST be returned before any FETCH responses, as otherwise the client might get confused about how message numbers map to UIDs.

Note: A server that receives a mod-sequence smaller than <minmodseq>, where <minmodseq> is the value of the smallest expunged mod-sequence it remembers minus one, MUST behave as if it was requested to report all expunged messages from the provided UID set parameter.

Example 1: Without the VANISHED UID FETCH modifier, a CONDSTORE-aware client needs to issue separate commands to learn of flag changes and expunged messages since the last synchronization:

```
C: s100 UID FETCH 300:500 (FLAGS) (CHANGEDSINCE 12345)
S: * 1 FETCH (UID 404 MODSEQ (65402) FLAGS (\Seen))
S: * 2 FETCH (UID 406 MODSEQ (75403) FLAGS (\Deleted))
S: * 4 FETCH (UID 408 MODSEQ (29738) FLAGS ($NoJunk
    $AutoJunk $MDNSent))
S: s100 OK FETCH completed
C: s101 UID SEARCH 300:500
S: * SEARCH 404 406 407 408 410 412
S: s101 OK search completed
```

Where 300 and 500 are the lowest and highest UIDs from client's cache. The second SEARCH response tells the client that the messages with UIDs 407, 410, and 412 are still present, but their flags haven't changed since the specified modification sequence.

Using the VANISHED UID FETCH modifier, it is sufficient to issue only a single command:

```
C: s100 UID FETCH 300:500 (FLAGS) (CHANGEDSINCE 12345
    VANISHED)
S: * VANISHED (EARLIER) 300:310,405,411
S: * 1 FETCH (UID 404 MODSEQ (65402) FLAGS (\Seen))
S: * 2 FETCH (UID 406 MODSEQ (75403) FLAGS (\Deleted))
S: * 4 FETCH (UID 408 MODSEQ (29738) FLAGS ($NoJunk
    $AutoJunk $MDNSent))
S: s100 OK FETCH completed
```

3.2.7. EXPUNGE Command

Arguments: none

Responses: untagged responses: EXPUNGE or VANISHED

Result: OK - expunge completed
NO - expunge failure: can't expunge (e.g., permission denied)
BAD - command unknown or arguments invalid

This section updates the definition of the EXPUNGE command described in Section 6.4.3 of [RFC3501].

The EXPUNGE command permanently removes all messages that have the \Deleted flag set from the currently selected mailbox. Before returning an OK to the client, those messages that are removed are reported using a VANISHED response or EXPUNGE responses.

If the server is capable of storing modification sequences for the selected mailbox, it MUST increment the per-mailbox mod-sequence if at least one message was permanently removed due to the execution of the EXPUNGE command. For each permanently removed message, the server MUST remember the incremented mod-sequence and corresponding UID. If at least one message got expunged and QRESYNC was enabled, the server MUST send the updated per-mailbox modification sequence using the HIGHESTMODSEQ response code (see Section 3.1.2.1) in the tagged OK response.

```
Example:    C: A202 EXPUNGE
            S: * 3 EXPUNGE
            S: * 3 EXPUNGE
            S: * 5 EXPUNGE
            S: * 8 EXPUNGE
            S: A202 OK [HIGHESTMODSEQ 20010715194045319] expunged
```

Note: In this example the client hasn't enabled QRESYNC, so the server is still using untagged EXPUNGE responses. Note that the presence of HIGHESTMODSEQ response code is optional in this case. If the selected mailbox returned NOMODSEQ, the HIGHESTMODSEQ response code will be absent. In this example, messages 3, 4, 7, and 11 had the \Deleted flag set. The first "* 3 EXPUNGE" reports message # 3 as expunged. The second "* 3 EXPUNGE" reports message # 4 as expunged (the message number got decremented due to the previous EXPUNGE response). See the description of the EXPUNGE response in [RFC3501] for further explanation.

Once the client enables QRESYNC, the the server will always send VANISHED responses instead of EXPUNGE responses for mailboxes that support storing of modification sequences, so the previous example might look like this:

```
   Example:    C: B202 EXPUNGE
               S: * VANISHED 405,407,410,425
               S: B202 OK [HIGHESTMODSEQ 20010715194045319] expunged
```

Here messages with message numbers 3, 4, 7, and 11 have respective UIDs 405, 407, 410, and 425.

3.2.8. CLOSE Command

Arguments: none

Responses: no specific responses for this command

Result: OK - close completed, now in authenticated state
BAD - command unknown or arguments invalid

This section updates the definition of the CLOSE command described in Section 6.4.2 of [RFC3501].

The CLOSE command permanently removes all messages that have the \Deleted flag set from the currently selected mailbox, and returns to the authenticated state from the selected state. No untagged EXPUNGE (or VANISHED) responses are sent.

If the server is capable of storing modification sequences for the selected mailbox, it MUST increment the per-mailbox mod-sequence if at least one message was permanently removed due to the execution of the CLOSE command. For each permanently removed message, the server MUST remember the incremented mod-sequence and corresponding UID. The server MUST NOT send the updated per-mailbox modification sequence using the HIGHESTMODSEQ response code (see Section 3.1.2.1) in the tagged OK response, as this might cause loss of synchronization on the client.

```
   Example:    C: A202 CLOSE
               S: A202 OK done
```

3.2.9. UID EXPUNGE Command

Arguments: message set

Responses: untagged responses: EXPUNGE or VANISHED

```
Result: OK - expunge completed
        NO - expunge failure: can't expunge (e.g., permission denied)
        BAD - command unknown or arguments invalid
```

This section updates the definition of the UID EXPUNGE command described in Section 2.1 of [UIDPLUS]. Servers that implement both [UIDPLUS] and QRESYNC extensions must implement UID EXPUNGE as described in this section.

The UID EXPUNGE command permanently removes from the currently selected mailbox all messages that both have the \Deleted flag set and have a UID that is included in the specified message set. If a message either does not have the \Deleted flag set or has a UID that is not included in the specified message set, it is not affected.

This command is particularly useful for disconnected mode clients. By using UID EXPUNGE instead of EXPUNGE when resynchronizing with the server, the client can avoid inadvertently removing any messages that have been marked as \Deleted by other clients between the time that the client was last connected and the time the client resynchronizes.

Before returning an OK to the client, those messages that are removed are reported using a VANISHED response or EXPUNGE responses.

If the server is capable of storing modification sequences for the selected mailbox, it MUST increment the per-mailbox mod-sequence if at least one message was permanently removed due to the execution of the UID EXPUNGE command. For each permanently removed message, the server MUST remember the incremented mod-sequence and corresponding UID. If at least one message got expunged and QRESYNC was enabled, the server MUST send the updated per-mailbox modification sequence using the HIGHESTMODSEQ response code (see Section 3.1.2.1) in the tagged OK response.

```
Example:    C: . UID EXPUNGE 3000:3002
            S: * 3 EXPUNGE
            S: * 3 EXPUNGE
            S: * 3 EXPUNGE
            S: . OK [HIGHESTMODSEQ 20010715194045319] Ok
```

Note: In this example the client hasn't enabled QRESYNC, so the server is still using untagged EXPUNGE responses instead of VANISHED responses. Note that the presence of HIGHESTMODSEQ response code is optional. If the selected mailbox returned NOMODSEQ, the HIGHESTMODSEQ response code will be absent. In this example, at least messages with message numbers 3, 4, and 5 (UIDs 3000 to 3002) had the \Deleted flag set. The first "* 3 EXPUNGE" reports message # 3 as expunged. The second "* 3 EXPUNGE" reports message # 4 as expunged (the message number got decremented due to the previous EXPUNGE response). See the description of the EXPUNGE response in [RFC3501] for further explanation.

3.2.10. VANISHED Response

Contents: an optional EARLIER tag

list of UIDs

The VANISHED response reports that the specified UIDs have been permanently removed from the mailbox. This response is similar to the EXPUNGE response [RFC3501]; however, it can return information about multiple messages, and it returns UIDs instead of message numbers. The first benefit saves bandwidth, while the second is more convenient for clients that only use UIDs to access the IMAP server.

The VANISHED response has the same restrictions on when it can be sent as does the EXPUNGE response (see below).

The VANISHED response has two forms. The first form contains the EARLIER tag, which signifies that the response was caused by a UID FETCH (VANISHED) or a SELECT/EXAMINE (QRESYNC) command. This response is sent if the UID set parameter to the UID FETCH (VANISHED) command includes UIDs of messages that are no longer in the mailbox. When the client sees a VANISHED EARLIER response, it MUST NOT decrement message sequence numbers for each successive message in the mailbox.

The second form doesn't contain the EARLIER tag and is described below. Once a client has issued "ENABLE QRESYNC" (and the server has positively responded to that command with the untagged ENABLED response containing QRESYNC), the server MUST use the VANISHED response without the EARLIER tag instead of the EXPUNGE response for

all mailboxes that don't return NOMODSEQ when selected. The server continues using VANISHED in lieu of EXPUNGE for the duration of the connection. In particular, this affects the EXPUNGE [RFC3501] and UID EXPUNGE [UIDPLUS] commands, as well as messages expunged in other connections. Such a VANISHED response MUST NOT contain the EARLIER tag.

A VANISHED response sent because of an EXPUNGE or UID EXPUNGE command or because messages were expunged in other connections (i.e., the VANISHED response without the EARLIER tag) also decrements the number of messages in the mailbox; it is not necessary for the server to send an EXISTS response with the new value. It also decrements message sequence numbers for each successive message in the mailbox (see the example at the end of this section).

Note that a VANISHED response without the EARLIER tag (i.e. a VANISHED response caused by EXPUNGE, UID EXPUNGE, or messages expunged in other connections) MUST only refer to a message which was visible to the client in the current session at the time the VANISHED response is sent. That is, servers MUST NOT send UIDs for previously expunged messages, or messages which were not announced to the client via EXISTS. This means that each UID listed in a VANISHED response results in the client decrementing the message count by one. This is required to prevent a possible race condition where new arrivals for which the UID is not yet known by the client are immediately expunged.

Because clients handle the two different forms of the VANISHED response differently, servers MUST NOT report UIDs resulting from a UID FETCH (VANISHED) or a SELECT/EXAMINE (QRESYNC) in the same VANISHED response as UIDs of messages expunged now (i.e., messages expunged in other connections). Instead, the server MUST send separate VANISHED responses: one with the EARLIER tag and one without.

A VANISHED response MUST NOT be sent when no command is in progress, nor while responding to a FETCH, STORE, or SEARCH command. This rule is necessary to prevent a loss of synchronization of message sequence numbers between client and server. A command is not "in progress" until the complete command has been received; in particular, a command is not "in progress" during the negotiation of command continuation.

Note: UID FETCH, UID STORE, and UID SEARCH are different commands from FETCH, STORE, and SEARCH. A VANISHED response MAY be sent during a UID command. However, the VANISHED response MUST NOT be sent during a UID SEARCH command that contains message numbers in the search criteria.

The update from the VANISHED response MUST be recorded by the client.

Example: Let's assume that there is the following mapping between message numbers and UIDs in the currently selected mailbox (here "X" marks messages with the \Deleted flag set, and "x" represents UIDs which are not relevant for the example):

```
Message numbers:   1    2    3    4    5  6   7  8  9 10  11
UIDs:              x  504  505  507  508  x 510  x  x  x 625
\Deleted messages:          X    X          X            X
```

In the presence of the extension defined in this document:

```
C: A202 EXPUNGE
S: * VANISHED 505,507,510,625
S: A202 OK EXPUNGE completed
```

Without the QRESYNC extension, the same example might look like:

```
C: A202 EXPUNGE
S: * 3 EXPUNGE
S: * 3 EXPUNGE
S: * 5 EXPUNGE
S: * 8 EXPUNGE
S: A202 OK EXPUNGE completed
```

(Continuing previous example) If subsequently messages with UIDs 504 and 508 got marked as \Deleted:

```
C: A210 EXPUNGE
S: * VANISHED 504,508
S: A210 OK EXPUNGE completed
```

i.e., the last VANISHED response only contains UIDs of messages expunged since the previous VANISHED response.

3.2.11. CLOSED Response Code

The CLOSED response code has no parameters. A server implementing the extension defined in this document MUST return the CLOSED response code when the currently selected mailbox is closed implicitly using the SELECT/EXAMINE command on another mailbox. The CLOSED response code serves as a boundary between responses for the previously opened mailbox (which was closed) and the newly selected mailbox: all responses before the CLOSED response code relate to the mailbox that was closed, and all subsequent responses relate to the newly opened mailbox.

There is no need to return the CLOSED response code on completion of the CLOSE or the UNSELECT [UNSELECT] command (or similar) whose purpose is to close the currently selected mailbox without opening a new one.

4. Long Command Lines

This document updates recommended line length limits specified in Section 3.2.1.5 of [RFC2683]. While the advice in the first paragraph of that section still applies ("use compact message/UID set representations"), the 1000 octet limit suggested in the second paragraph turned out to be quite problematic when the CONDSTORE and/or QRESYNC extension is used. The updated recommendation is as follows: a client should limit the length of the command lines it generates to approximately 8192 octets (including all quoted strings but not including literals).

5. QRESYNC Server Implementation Considerations

This section describes a minimalist implementation, a moderate implementation, and an example of a full implementation.

5.1. Server Implementations That Don't Store Extra State

Strictly speaking, a server implementation that doesn't remember mod-sequences associated with expunged messages can be considered compliant with this specification. Such implementations return all expunged messages specified in the UID set of the UID FETCH (VANISHED) command every time, without paying attention to the specified CHANGEDSINCE mod-sequence. Such implementations are discouraged, as they can end up returning VANISHED responses that are bigger than the result of a UID SEARCH command for the same UID set.

Clients that use the message sequence match data can reduce the scope of this VANISHED response substantially in the typical case where expunges have not happened, or happen only toward the end of the mailbox.

5.2. Server Implementations Storing Minimal State

A server that stores the HIGHESTMODSEQ value at the time of the last EXPUNGE can omit the VANISHED response when a client provides a MODSEQ value that is equal to, or higher than, the current value of this datum, that is, when there have been no EXPUNGEs.

A client providing message sequence match data can reduce the scope as above. In the case where there have been no expunges, the server can ignore this data.

5.3. Additional State Required on the Server

When compared to the CONDSTORE extension, QRESYNC requires servers to store additional state associated with expunged messages. Note that implementations are not required to store this state in persistent storage; however, use of persistent storage is advisable.

One possible way to correctly implement the extension described in this document is to store a queue of <UID set, mod-sequence> pairs. <UID set> can be represented as a sequence of <min UID, max UID> pairs.

When messages are expunged, one or more entries are added to the queue tail.

When the server receives a request to return messages expunged since a given mod-sequence, it will search the queue from the tail (i.e., going from the highest expunged mod-sequence to the lowest) until it sees the first record with a mod-sequence less than or equal to the given mod-sequence or it reaches the head of the queue.

Note that indefinitely storing information about expunged messages can cause storage and related problems for an implementation. In the worst case, this could result in almost 64Gb of storage for each IMAP mailbox. For example, consider an implementation that stores <min UID, max UID, mod-sequence> triples for each range of messages expunged at the same time. Each triple requires 16 octets: 4 octets for each of the two UIDs, and 8 octets for the mod-sequence. Assume that there is a mailbox containing a single message with a UID of 2**32-1 (the maximum possible UID value), where messages had previously existed with UIDs starting at 1, and have been expunged one at a time. For this mailbox alone, storage is required for the triples <1, 1, modseq1>, <2, 2, modseq2>, ..., <2**32-2, 2**32-2, modseq4294967294>.

Hence, implementations are encouraged to adopt strategies to protect against such storage problems, such as limiting the size of the queue used to store mod-sequences for expunged messages and "expiring" older records when this limit is reached. When the selected implementation-specific queue limit is reached, the oldest record(s) are deleted from the queue (note that such records are located at the queue head). For all such "expired" records, the server needs to store a single mod-sequence, which is the highest mod-sequence for all "expired" expunged messages.

Note that if the client provides the message sequence match data, this can heavily reduce the data cost of sending a complete set of missing UIDs; thus, reducing the problems for clients if a server is

unable to persist much of this queue. If the queue contains data back to the requested mod-sequence, this data can be ignored.

Also, note that if the UIDVALIDITY of the mailbox changes or if the mailbox is deleted, then any state associated with expunged messages doesn't need to be preserved and SHOULD be deleted.

6. Updated Synchronization Sequence

This section updates the description of optimized synchronization in Section 6.1 of the [IMAP-DISC].

An advanced disconnected mail client SHOULD use the QRESYNC and CONDSTORE extensions when they are supported by the server. The client uses the value from the HIGHESTMODSEQ OK response code received on mailbox opening to determine if it needs to resynchronize. Once the synchronization is complete, it MUST cache the received value (unless the mailbox UIDVALIDITY value has changed; see below). The client MUST update its copy of the HIGHESTMODSEQ value whenever the server sends a subsequent HIGHESTMODSEQ OK response code.

After completing a full synchronization, the client MUST also take note of any unsolicited MODSEQ FETCH data items and HIGHESTMODSEQ response codes received from the server. Whenever the client receives a tagged response to a command, it checks the received unsolicited responses to calculate the new HIGHESTMODSEQ value. If the HIGHESTMODSEQ response code is received, the client MUST use it even if it has seen higher mod-sequences. Otherwise, the client calculates the highest value among all MODSEQ FETCH data items received since the last tagged response. If this value is bigger than the client's copy of the HIGHESTMODSEQ value, then the client MUST use this value as its new HIGHESTMODSEQ value.

Example:

```
C: A150 STORE 1:2 (UNCHANGEDSINCE 96) +FLAGS.SILENT \Seen
S: * 1 FETCH (UID 6 MODSEQ (103))
S: * 2 FETCH (UID 7 MODSEQ (101))
S: * OK [HIGHESTMODSEQ 99] VANISHED reply with MODSEQ 100 is delayed
S: A150 OK [MODIFIED 3] done

C: A151 STORE 3 +FLAGS.SILENT \Seen
S: * 3 FETCH (UID 8 MODSEQ (104))
S: A151 OK [HIGHESTMODSEQ 99] Still delaying VANISHED

C: A152 NOOP
S: * VANISHED 8
S: A153 OK [HIGHESTMODSEQ 104] done
```

Note: It is not safe to update the client's copy of the HIGHESTMODSEQ value with a MODSEQ FETCH data item value as soon as it is received because servers are not required to send MODSEQ FETCH data items in increasing modseqence order. Some commands may also delay EXPUNGE (or VANISHED) replies with smaller mod-sequences. These can lead to the client missing some changes in case of connectivity loss.

When opening the mailbox for synchronization, the client uses the QRESYNC parameter to the SELECT/EXAMINE command. The QRESYNC parameter is followed by the UIDVALIDITY and mailbox HIGHESTMODSEQ values, as known to the client. It can be optionally followed by the set of UIDs, for example, if the client is only interested in partial synchronization of the mailbox. The client may also transmit a list containing its knowledge of message numbers.

If the SELECT/EXAMINE command is successful, the client compares UIDVALIDITY as described in step d)1) in Section 3 of the [IMAP-DISC]. If the cached UIDVALIDITY value matches the one returned by the server and the server also returns the HIGHESTMODSEQ response code, then the server reports expunged messages and returns flag changes for all messages specified by the client in the UID set parameter (or for all messages in the mailbox, if the client omitted the UID set parameter). At this point, the client is synchronized, except for maybe the new messages.

If upon a successful SELECT/EXAMINE (QRESYNC) command the client receives a NOMODSEQ OK untagged response (instead of the HIGHESTMODSEQ response code), it MUST remove the last known HIGHESTMODSEQ value from its cache and follow the more general instructions in Section 3 of the [IMAP-DISC].

At this point, the client is in sync with the server regarding old messages. This client can now fetch information about new messages (if requested by the user).

Step d) ("Server-to-client synchronization") in Section 4 of the [IMAP-DISC] in the presence of the QRESYNC & CONDSTORE extensions is amended as follows:

d) "Server-to-client synchronization" -- for each mailbox that requires synchronization, do the following:

1a) Check the mailbox UIDVALIDITY (see Section 4.1 of the [IMAP-DISC] for more details) after issuing SELECT/EXAMINE (QRESYNC) command.

If the UIDVALIDITY value returned by the server differs, the client MUST

* empty the local cache of that mailbox;

* "forget" the cached HIGHESTMODSEQ value for the mailbox;

* remove any pending "actions" which refer to UIDs in that mailbox. Note, this doesn't affect actions performed on client generated fake UIDs (see Section 5 of the [IMAP-DISC]);

2) Fetch the current "descriptors";

I) Discover new messages.

3) Fetch the bodies of any "interesting" messages that the client doesn't already have.

Example: The UIDVALIDITY value is the same, but the HIGHESTMODSEQ value has changed on the server while the client was offline:

```
C: A142 SELECT INBOX (QRESYNC (3857529045 20010715194032001 1:198))
S: * 172 EXISTS
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * OK [UIDNEXT 201] Predicted next UID
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S: * OK [HIGHESTMODSEQ 20010715194045007] Highest
      mailbox mod-sequence
S: * VANISHED (EARLIER) 1:5,7:8,10:15
S: * 2 FETCH (UID 6 MODSEQ (20010715205008000)
    FLAGS (\Deleted))
S: * 5 FETCH (UID 9 MODSEQ (20010715195517000)
    FLAGS ($NoJunk $AutoJunk $MDNSent))
   ...
S: A142 OK [READ-WRITE] SELECT completed
```

7. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

Non-terminals referenced but not defined below are as defined by [RFC5234], [RFC3501], or [RFC4466].

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" / "QRESYNC"

status-att          =/ "HIGHESTMODSEQ"
                       ;; extends non-terminal defined in [RFC3501].

status-att-val      =/ "HIGHESTMODSEQ" SP mod-sequence-valzer
                       ;; extends non-terminal defined in [RFC4466].
                       ;; Value 0 denotes that the mailbox doesn't
                       ;; support persistent mod-sequences
                       ;; as described in Section 3.1.2.2

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 [RFC4466].

   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, SORT and THREAD.

   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 sequence-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 [RFC3501], 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 63-bit integer
                         ;; (mod-sequence)
                         ;; (1 <= n <= 9,223,372,036,854,775,807)

   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 [RFC4466].

   condstore-param     = "CONDSTORE"

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

   sort-data           = "SORT" [1*(SP nz-number) SP
                         search-sort-mod-seq]
                         ; Updates SORT response from RFC 5256

   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 [RFC3501].

   attr-flag-keyword   = atom

   select-param        =  "QRESYNC" SP "(" uidvalidity SP
                       mod-sequence-value [SP known-uids]
                       [SP seq-match-data] ")"
                       ;; conforms to the generic select-param
```

```
                    ;; syntax defined in [RFC4466]

   seq-match-data   =  "(" known-sequence-set SP known-uid-set ")"

   uidvalidity      =  nz-number

   known-uids       =  sequence-set
                       ;; sequence of UIDs, "*" is not allowed

   known-sequence-set  =  sequence-set
                       ;; set of message numbers corresponding to
                       ;; the UIDs in known-uid-set, in ascending order.
                       ;; * is not allowed.

   known-uid-set    =  sequence-set
                       ;; set of UIDs corresponding to the messages in
                       ;; known-sequence-set, in ascending order.
                       ;; * is not allowed.

   message-data     =/ expunged-resp

   expunged-resp    =  "VANISHED" [SP "(EARLIER)"] SP known-uids

   rexpunges-fetch-mod =  "VANISHED"
                       ;; VANISHED UID FETCH modifier conforms
                       ;; to the fetch-modifier syntax
                       ;; defined in [RFC4466].  It is only
                       ;; allowed in the UID FETCH command.

   resp-text-code   =/ "CLOSED"
```

8. Security Considerations

As always, it is important to thoroughly test clients and servers implementing QRESYNC, as it changes how the server reports expunged messages to the client.

It is believed that the CONDSTORE or the QRESYNC extensions don't raise any new security concerns that are not already discussed in [RFC3501]. However, the availability of CONDSTORE 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.

9. 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 and QRESYNC IMAP capabilities. IANA is requested to update references for both extensions to point to this document.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2683] Leiba, B., "IMAP4 Implementation Recommendations", RFC 2683, September 1999.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC4466] Melnikov, A. and C. Daboo, "Collected Extensions to IMAP4 ABNF", RFC 4466, April 2006.

[RFC5161] Gulbrandsen, A. and A. Melnikov, "The IMAP ENABLE Extension", RFC 5161, March 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5256] Crispin, M. and K. Murchison, "Internet Message Access Protocol - SORT and THREAD Extensions", RFC 5256, June 2008.

[RFC5464] Daboo, C., "The IMAP METADATA Extension", RFC 5464, February 2009.

[UIDPLUS] Crispin, M., "Internet Message Access Protocol (IMAP) - UIDPLUS extension", RFC 4315, December 2005.

10.2. Informative References

[IMAP-DISC] Melnikov, A., Ed., "Synchronization Operations For Disconnected Imap4 Clients", RFC 4549, June 2006.

[NTP] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC2180] Gahrns, M., "IMAP4 Multi-Accessed Mailbox Practice", RFC 2180, July 1997.

[RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, December 2005.

[RFC4731] Melnikov, A. and D. Cridland, "IMAP4 Extension to SEARCH Command for Controlling What Kind of Information Is Returned", RFC 4731, November 2006.

[RFC5257] Daboo, C. and R. Gellens, "Internet Message Access Protocol - ANNOTATE Extension", RFC 5257, June 2008.

[RFC5267] Cridland, D. and C. King, "Contexts for IMAP4", RFC 5267, July 2008.

[RFC6851] Gulbrandsen, A. and N. Freed, "Internet Message Access Protocol (IMAP) - MOVE Extension", RFC 6851, January 2013.

[UNSELECT] Melnikov, A., "Internet Message Access Protocol (IMAP) UNSELECT command", RFC 3691, February 2004.

Appendix A. Changes since RFC 4551

Changed mod-sequences to be unsigned 63-bit values (instead of unsigned 64-bit values).

Fixed errata 3401 ("several typos in UNCHANGEDSINCE spelling"), 3506 ("invalid ABNF for the MODIFIED response code") and 3509 ("correction to an example").

Clarified that returning of HIGHESTMODSEQ/NOMODSEQ response codes is only required once a CONDSTORE enabling command was issued.

Clarified that if multiple mod-sequences (for different metadata items) are associated with a message, then all of them affecting a particular STORE UNCHANGEDSINCE must be checked.

Updated references.

Editorial corrections.

Appendix B. Changes since RFC 5162

Changed mod-sequences to be unsigned 63-bit values (instead of unsigned 64-bit values).

Addressed erratum #1365: http://www.rfc-editor.org/errata_search.php?eid=1365

Addressed erratum #1807: http://www.rfc-editor.org/errata_search.php?eid=1807

Addressed erratum #1808: http://www.rfc-editor.org/errata_search.php?eid=1808

Addressed erratum #1809: http://www.rfc-editor.org/errata_search.php?eid=1809

Addressed erratum #1810: http://www.rfc-editor.org/errata_search.php?eid=1810

Addressed erratum #3322: http://www.rfc-editor.org/errata_search.php?eid=3322

Clarified that ENABLE QRESYNC CONDSTORE and ENABLE CONDSTORE QRESYNC are equivalent.

Changed the requirement to return VANISHED from SHOULD to MUST as per the mailing list discussion. The only exception is for mailboxes that return NOMODSEQ response code when they are selected.

Specified that IMAP SETMETADATA changes update per-mailbox HIGHESTMODSEQ.

Clarified that per-message annotations are also considered "metadata".

Fixed some examples to report data that match requirements specified in the document.

Clarified some text and made some requirements normative. Also corrected a couple of SHOULDs to be MUSTs.

Updated references.

Editorial corrections.

Appendix C. Acknowledgements

Thank you to Steve Hole for co-editing RFC 4551.

In this revision of the document the authors also acknowledge the feedback provided by Timo Sirainen, Jan Kundrat, Pete Maclean, Eliot Lear, Chris Newman, Claudio Allocchio, Michael Slusarz, Bron Gondwana, Hoa V. DINH and Mark Crispin.

See also RFC 4551 for the list of people who contributed to earlier version of this RFC.

Authors' Addresses

Alexey Melnikov
Isode Ltd
5 Castle Business Village
36 Station Road
Hampton, Middlesex TW12 2BX
UK

Email: Alexey.Melnikov@isode.com

Dave Cridland
Arcode Inc
4304 East West Highway
Bethesda, MD 20814
US

Email: dcridland@arcode.com