Network Working Group   A. Melnikov
Internet-Draft   D. Cridland
Intended status: Standards Track   Isode Ltd
Expires: December 13, 2007   C. Wilson
                                                                Nokia
                                                        June 11, 2007

IMAP4 Extensions for Quick Mailbox Resynchronization
draft-ietf-lemonade-reconnect-client-05.txt

Status of this Memo

Copyright Notice

Abstract

   This document defines an IMAP4 extension, which gives an IMAP client
   the ability to quickly resynchronize any previously opened mailbox as
   part of the SELECT command, without the need for server-side state or
   additional client round-trips. This extension also introduces a new
   response that allows for a more compact representation for a list of
   expunged messages.

Changes since draft-ietf-lemonade-reconnect-client-04.txt

   o Clarified handling of omitted list of known UIDs.

   o Clarified interaction with the CONDSTORE extension.

   o Renamed TAG to EARLIER and removed the associated tag parameter.

   o Made QRESYNC extension depend on ENABLE.

   o Removed text about client emulating UID EXPUNGE in absence of
      UIDPLUS.

   o Added CLOSED response code.

Changes since draft-ietf-lemonade-reconnect-client-03.txt

   o Fixed several typos reported by Randy Gellens.

   o Fixed a couple of errors in examples and text reported by Dan
      Karp.

Changes since draft-ietf-lemonade-reconnect-client-02.txt

   o Fixed description of the synchronization sequence to properly
      describe how HIGHESTMODSEQ is used.

   o Fixed a couple of errors in ABNF.

Changes since draft-ietf-lemonade-reconnect-client-01.txt

   o Folded the EXPUNGED extension
      (draft-melnikov-imap-expunged-02.txt) into this document. Updated
      mailbox synchronization instructions.

   o Added UID sequence number matching.

   o Clarified how NOMODSEQ response affects this extension.

   o Other minor editorial changes and fixes.

Changes since draft-ietf-lemonade-reconnect-client-00.txt

   o Changed server behavior when the specified UIDVALIDITY doesn't
      match the current. This allows the client to chose how to proceed
      after that.

   o If client's UIDVALIDITY doesn't match server's, the server will
      not return any flags anymore.

   o Clarified that SELECT (QRESYNC) is a CONDSTORE-enabling command.

   o Other minor editorial changes and fixes.

Table of Contents

1. 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.

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

   [[anchor2: Editorial comments and questions are marked like this.]]

2. Introduction and Overview

   The [CONDSTORE] extension gives a disconnected client 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. This document defines an extension to [CONDSTORE]
   that allows a reconnecting client to perform full resynchronization,
   including discovery of expunged messages, in a single round-trip.
   This extension also introduces a new response VANISHED that allows
   for a more compact representation for a list of expunged messages.

   This extension 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, without forcing the user to experience long delay
   and pay big bills for 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.

   [[anchor4: Note to RFC editor: Please change the capability name
   everywhere in the document from "X-DRAFT-W05-QRESYNC" to "QRESYNC".]]

   The quick resync IMAP extension is present if an IMAP4 server returns
   "X-DRAFT-W05-QRESYNC" as one of the supported capabilities to the

CAPABILITY command.

Servers supporting this extension MUST implement and advertise
support for the [ENABLE] IMAP extension. Also, the presence of the
"X-DRAFT-W05-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 X-DRAFT-W05-QRESYNC" and "ENABLE X-DRAFT-W05-QRESYNC
CONDSTORE" (which are "CONDSTORE enabling commands", as defined in
[CONDSTORE], and have identical results), but there is no requirement
for a compliant server to support "ENABLE CONDSTORE" by itself. The
"ENABLE X-DRAFT-W05-QRESYNC"/"ENABLE X-DRAFT-W05-QRESYNC CONDSTORE"
command also tells the server that it SHOULD start sending VANISHED
responses (see Section 3.6) instead of EXPUNGE responses. This
change remains in effect until the connection is closed.

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

This document 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 has sent 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 or CLOSE; the
server MUST associate the incremented mod-sequence with the UIDs of
the expunged messages.

A client which supports CONDSTORE but not this extension might
resynchronize a mailbox and discover that its HIGHESTMODSEQ has
increased from the value cached by the client. If the increase is
due only 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 which supports
CONDSTORE but not this extension might incur a penalty of an unneeded
round-trip when resynchronizing some mailboxes (those which have had
messages expunged but no flag changes since the last
synchronization).

This extra round-trip is only incurred by clients that supports
CONDSTORE but not this extension, 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 this extension.

[3](#). **IMAP Protocol Changes**

[3.1](#). **QRESYNC parameter to SELECT/EXAMINE**

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

o the last known UIDVALIDITY,

o the last known modification sequence

o the optional set of known UIDs

o an optional parenthesized list of known sequence ranges and their
   corresponding UIDs.

The parameter acts as a CONDSTORE enabling command, as defined in
[CONDSTORE]. In other words, the use of the QRESYNC parameter
implies the CONDSTORE parameter. The QRESYNC parameter also tells
the server that it SHOULD start sending VANISHED responses (see
Section 3.6) instead of EXPUNGE responses. This change remains in
effect until the connection is closed.

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 server MUST send
all untagged responses as specified in Section 6.3.1/6.3.2 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 behaviour 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]
        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 the OK untagged response including the
NOMODSEQ response code with every successful SELECT or EXAMINE
command, as described in [CONDSTORE]. Such 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.

However, whether the server returns the HIGHESTMODSEQ or the NOMODSEQ
response code, the QRESYNC parameter still enables the use of the
VANISHED response in lieu of the EXPUNGE response Section 3.6.

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 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 provided 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)

Example:

```
C: A03 SELECT INBOX (QRESYNC (67890007
   90060115194045000 41,43:211,214:541))

S: * OK [CLOSED]

S: * 314 EXISTS

S: * 15 RECENT

S: * OK [UIDVALIDITY 67890007] UIDVALIDITY

S: * OK [UIDNEXT 567] Predicted next UID

S: * OK [HIGHESTMODSEQ 90060115205545359]

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: * 49 FETCH (UID 117 FLAGS (\Seen \Answered))

S: * 50 FETCH (UID 119 FLAGS (\Draft $MDNSent))

S: ...

S: * 100 FETCH (UID 541 FLAGS (\Seen $Forwarded))

S: * VANISHED 41,43:116,118,120:211,214:540

S: A03 OK [READ-WRITE] mailbox selected
```

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:

```
C: A04 SELECT INBOX (QRESYNC (67890007
   90060115194045000 1:29997))

S: * 10003 EXISTS

S: * 5 RECENT

S: * OK [UIDVALIDITY 67890007] UIDVALIDITY

S: * OK [UIDNEXT 30013] Predicted next UID

S: * OK [HIGHESTMODSEQ 90060115205545359]

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: * 9889 FETCH (UID 29667 FLAGS (\Seen \Answered))

S: * 9890 FETCH (UID 29670 FLAGS (\Draft $MDNSent))
```

```
   S: ...

   S: * 9999 FETCH (UID 29997 FLAGS (\Seen $Forwarded))

   S: * VANISHED 1:2,4:5,7:8,10:11,13:14 [...]
      29998:29999,30001:30002,30004:30005,30007:30008

   S: A04 OK [READ-WRITE] mailbox selected
```

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: * 5 RECENT

   S: * OK [UIDVALIDITY 67890007] UIDVALIDITY

   S: * OK [UIDNEXT 30013] Predicted next UID

   S: * OK [HIGHESTMODSEQ 90060115205545359]

   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: * 9889 FETCH (UID 29667 FLAGS (\Seen \Answered))

   S: * 9890 FETCH (UID 29670 FLAGS (\Draft $MDNSent))

   S: ...

   S: * 9999 FETCH (UID 29997 FLAGS (\Seen $Forwarded))

   S: * VANISHED 29998:29999,30001:30002,30004:30005,30007:30008

   S: B04 OK [READ-WRITE] mailbox selected
```

3.2. **VANISHED UID FETCH modifier**

   [IMAPABNF] 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.

   The VANISHED UID FETCH modifier MUST only be specified together with
   the CHANGEDSINCE UID FETCH modifier.

   The VANISHED UID FETCH modifier instructs the server to report those
   messages from the UID set parameter that have been expunged and whose
   associated modsequence is larger than the specified modsequence.
   That is, the client requests to be informed of messages from the
   specified set that were expunged since the specified modsequence.
   Note that the modsequence(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 3.6, which MUST contain the EARLIER tag.

   Note: a server that receives a modsequence smaller than any of the
   expunged modsequence it remembers about MUST behave as if it was
   requested to report all expunged messages from the provided UID set
   parameter.

   The VANISHED UID FETCH modifier also instructs the server to replace
   all further EXPUNGE responses with VANISHED responses. The server
   MUST do this until the connection is closed.

   Example 1: Without the VANISHED UID FETCH modifier a CONDSTORE-aware
   client [CONDSTORE] 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: * 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: * VANISHED (EARLIER) 300:310,405,411
S: s100 OK FETCH completed
```

3.3. 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, the server MUST send the updated per-mailbox modification sequence using the HIGHESTMODSEQ response code (defined in [CONDSTORE]) 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, messages 3, 4, 7, and 11 had the \Deleted flag
set. See the description of the EXPUNGE response in [RFC3501] for
further explanation.

Note that once the VANISHED response is enabled on the connection 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.

3.4. 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. If
at least one message got expunged, the server MUST send the updated
per-mailbox modification sequence using the HIGHESTMODSEQ response
code (defined in [CONDSTORE]) in the tagged OK response.

Example: C: A202 CLOSE
             S: A202 OK [HIGHESTMODSEQ 20010715194045319] done

3.5. 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 X-DRAFT-W05-QRESYNC [[anchor11: RFC editor: change
before publication]] 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, the server MUST send the
updated per-mailbox modification sequence using the HIGHESTMODSEQ
response code (defined in [CONDSTORE]) 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, at least messages with message numbers 3, 4,
   and 5 (UIDs 3000 to 3002) had the \Deleted flag set. See the
   description of the EXPUNGE response in [RFC3501] for further
   explanation.

3.6. 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 which 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. Such
   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 used "(VANISHED)" with a UID FETCH or
   "(QRESYNC)" with SELECT/EXAMINE command, the server SHOULD use the
   VANISHED response without the EARLIER tag instead of the EXPUNGE
   response. The server SHOULD continue 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 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 and/or RECENT 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 caused by EXPUNGE/UID EXPUNGE/messages expunged
in other connections SHOULD only contain UIDs for messages expunged
since the last VANISHED/EXPUNGE response sent for the currently
opened mailbox or since the mailbox was opened. That is, servers
SHOULD NOT send UIDs for previously expunged messages, unless
explicitly requested to do so by the UID FETCH (VANISHED) command.

Note that client implementors must take care to properly decrement
the number of messages in the mailbox even if a server violates this
last SHOULD or repeats the same UID multiple times in the returned
UID set. In general this means that a client using this extension
should either avoid using message numbers entirely, or have a
complete map of UID-to-message mapping for the selected mailbox.

Because clients handle the two different forms of the VANISHED
response differently, servers MUST NOT report UIDs resulting from an
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 X-DRAFT-W05-QRESYNC [[anchor12: RFC editor: change before
publication]] 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.

3.7. 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.
There is no need to return this response code on completion of the
CLOSE or the UNSELECT [UNSELECT] command (or similar) who's purpose
is to close the currently selected mailbox without opening a new one.
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 which was closed and all subsequent responses

   relate to the newly opened mailbox.

4. Server implementation considerations

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

4.1. Server implementations that don't store extra state

   Strictly speaking, a server implementation that doesn't remember
   modsequences 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 modsequence. Such implementations are
   discouraged, as they can end up returning VANISHED responses bigger
   than the result of a UID SEARCH command for the same UID set.

   Clients which 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.

4.2. Server implementations storing minimal state

   A server which 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.

4.3. Additional state required on the server

   When compared to the [CONDSTORE] extension, this extension 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, modsequence> pairs.
   <UID set> can be represented as a sequence of <min UID, max UID>
   pairs.

   When messages are expunged, one or more entry is added to the queue

   tail.

   When the server receives a request to return expunged messages since
   a given modsequence, it will search the queue from the tail (i.e.
   going from the highest expunged modsequence to the lowest), until it
   sees the first record with a modsequence less than or equal to the
   given modsequence, 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, modsequence> 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 modsequence. Assume 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 modsequences 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 modsequence, which is the highest modsequence 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 modsequence, 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
   MUST be deleted as well.

5. 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 X-DRAFT-W05-

QRESYNC [[anchor18: RFC editor: change before publication]] 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.

The client MUST also take note of any MODSEQ FETCH data items
received from the server. Whenever the client receives a tagged
response to a command, it 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.

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. This can lead to the client missing
some changes in case of connectivity loss.

When opening the mailbox for synchronization the client uses 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/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 X-DRAFT-W05-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]);

             *   skip steps 1b and 2-II;

   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]
   S: * 2 FETCH (UID 6 MODSEQ (20010715205008000)
       FLAGS (\Deleted))
   S: * 5 FETCH (UID 9 MODSEQ (20010715195517000)
       FLAGS ($NoJunk $AutoJunk $MDNSent))
       ...
   S: * VANISHED 1:5,7:8,10:15
   S: A142 OK [READ-WRITE] SELECT completed
```

6. **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
   [RFC3501], [CONDSTORE] or [IMAPABNF].

   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          =/  "X-DRAFT-W05-QRESYNC"
                       ;; [[Note to RFC Editor: change before
                       ;; publication]]

   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 [IMAPABNF]

   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 [IMAPABNF].  It is only
                     ;; allowed in the UID FETCH command.

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

7. Security Considerations

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

Security considerations relevant to [CONDSTORE] are relevant to this
extension.

This document doesn't raise any new security concerns not already
raised by [CONDSTORE] or [RFC3501].

8. 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 X-DRAFT-W05-QRESYNC [[anchor22: The final

capability name will be chosen during AUTH48]] IMAP capability. IANA
is requested to add this capability to the registry.

9. Acknowledgments

Thanks to Steve Hole, Cyrus Daboo and Michael Wener for encouraging
creation of this document.

Valuable comments, both in agreement and in dissent, were received
from Timo Sirainen, Michael Wener, Randall Gellens, Arnt Gulbrandsen,
Chris Newman, Peter Coates, Mark Crispin, Elwyn Davies, Dan Karp and
Mike Zraly.

This document takes substantial text from [RFC3501] by Mark Crispin.

10. References

10.1. Normative References

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

[CONDSTORE]
            Melnikov, A. and S. Hole, "IMAP Extension for Conditional
            STORE Operation or Quick Flag Changes Resynchronization",
            RFC 4551, June 2006.

[ENABLE]   Gulbrandsen, A., "The IMAP ENABLE Extension",
            draft-gulbrandsen-imap-enable (work in progress),
            March 2007.

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

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

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

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

10.2. Informative References

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

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

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

   Email: dave.cridland@isode.com

   Corby Wilson
   Nokia
   5 Wayside Rd.
   Burlington, MA 01803
   USA

   Email: corby@computer.org