Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 19, 2007

A. Melnikov
Isode Ltd

October 16, 2006

IMAP4 extension for reporting expunged messages
draft-melnikov-imap-expunged-02.txt

Status of this Memo

Copyright Notice

Abstract

This document defines an IMAP extension, which gives a disconnected client ability to quickly learn about expunged messages. This extension also introduces a new response that allows for a more compact representation for a list of expunged messages.

Table of Contents

1. Conventions Used in this Document

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 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].

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 synchronize IMAP flag changes for previously seen messages. 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 quickly learn about expunged messages. This extension also introduces a new response VANISHED that allows for a more compact representation for a list of expunged messages.

The Expunged Messages Notification extension is present in any IMAP4 implementation which advertises "X-DRAFT-I02-EXPUNGED" [[anchor4: Change before publication]] as one of the supported capabilities in the CAPABILITY command response. Any server returning the capability MUST also support the The [CONDSTORE] extension.

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.

This change doesn't affect a client that only supports the CONDSTORE extension, however per-mailbox mod-sequence change due to expunges may force the client to send FETCH CHANGEDSINCE that will return no

data, thus forcing additional round-trip. [[anchor5: Is this Ok?]] [[anchor6: Should we have a separate per mailbox modseq (for expunged messages) just to address this issue?]]

3. IMAP Protocol Changes

3.1. REPORTEXPUNGES 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: REPORTEXPUNGES.

Note, that the REPORTEXPUNGES 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 REPORTEXPUNGES UID FETCH modifier MUST only be specified together with the CHANGEDSINCE UID FETCH modifier. [[anchor9: alternative: allow REPORTEXPUNGES by itself, this will mean report all expunged UIDs from the UID set specified in the UID FETCH.]]

The REPORTEXPUNGES 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 associated with these messages was incremented 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 TAG correlator.

The REPORTEXPUNGES 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 REPORTEXPUNGES 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 REPORTEXPUNGES UID FETCH modifier it is sufficient to issue only a single command:

```
C: s100 UID FETCH 300:500 (FLAGS) (CHANGEDSINCE 12345
    REPORTEXPUNGES)
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 (TAG "s100") 300:310,405,411
S: s100 OK FETCH completed
```

3.2. REPORTEXPUNGES Parameter to SELECT and EXAMINE

The X-DRAFT-I02-EXPUNGED extension defines a single optional select parameter, "REPORTEXPUNGES", which 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.

[[anchor11: Note that if we want to make the REPORTEXPUNGES parameter to SELECT/EXAMINE also behave as if UID FETCH (REPORTEXPUNGES) were specified, then we need to add 3 extra arguments to the REPORTEXPUNGES parameter (a la QRESYNC): the last known UIDVALIDITY, the last known modification sequence and the optional list of known UIDs.]]

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 Ok
```

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 Ok
```

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-I02-EXPUNGED 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.

If the server does not support the UIDPLUS capability, the client SHOULD fall back to using the STORE command to temporarily remove the \Deleted flag from messages it does not want to remove, then issuing the EXPUNGE command. Finally, the client SHOULD use the STORE command to restore the \Deleted flag on the messages in which it was temporarily removed.

Alternatively, the client MAY fall back to using just the EXPUNGE command, risking the unintended removal of some messages.

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: optional correlators

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 starts with an optional correlator. If it is present and contains the TAG correlator type, then the response is a result of a UID FETCH (REPORTEXPUNGES) command. Other correlators can be added in the future.

The VANISHED response is sent as a result of a UID FETCH (REPORTEXPUNGES) command, if the UID set parameter to the UID FETCH (REPORTEXPUNGES) command includes UIDs of messages that are no longer in the mailbox. Such VANISHED response MUST contain the TAG correlator.

Once a client has used "(REPORTEXPUNGES)" with a UID FETCH or SELECT/EXAMINE command, the server SHOULD use the VANISHED response 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 TAG correlator.

A VANISHED response sent because of an EXPUNGE or UID EXPUNGE command or because messages were expunged in other connections 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 (REPORTEXPUNGES) 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.

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-I02-EXPUNGED [[anchor15: 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.

4. Server implementation considerations

This section describes a poor implementation and an example of a good 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

(REPORTEXPUNGES) 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.

4.2. 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 would be 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 expunged messages expunged at once. 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.

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-I02-EXPUNGED and [CONDSTORE] extensions when they are supported by the server. The client MUST cache the value from HIGHESTMODSEQ OK response code received on mailbox opening and update it whenever the server sends MODSEQ FETCH data items.

If the client receives a NOMODSEQ OK untagged response instead of HIGHESTMODSEQ, it MUST remove the last known HIGHESTMODSEQ value from its cache and follow the more general instructions in section 3 of the [IMAP-DISC].

When the client opens the mailbox for synchronization it first 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, the client MUST compare the cached value of HIGHESTMODSEQ with the one returned by the server. If the cached HIGHESTMODSEQ value also matches the one returned by the server, then the client SHOULD NOT fetch flags for cached messages, as they haven't changed. If the value returned by the server is higher than the cached one, the client MAY use "SEARCH MODSEQ <cached-value>" to find all messages with flags changed since the last time the client was online and had the mailbox opened. Alternatively the client MAY use "FETCH 1:* (FLAGS) (CHANGEDSINCE <cached-value> REPORTEXPUNGES)". The latter operation combines reporting expunged messages, searching

for changed messages and fetching new information.

In all cases the client still needs to fetch information about new messages (if requested by the user). If the client has used SEARCH MODSEQ, it will also have to discover which messages have been expunged.

Step d) ("Server-to-client synchronization") in section 4 of the [IMAP-DISC] in the presence of the X-DRAFT-I02-EXPUNGED & 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) with SELECT/EXAMINE/STATUS. 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;

1b) Check the mailbox HIGHESTMODSEQ. If the cached value is the same as the one returned by the server, skip fetching message flags on step 2-II, i.e., the client only has to find out which messages got expunged.

2) Fetch the current "descriptors";

I) Discover new messages.

II) Discover changes to old messages and expunged messages using "UID FETCH 1:<lastseenuid> (FLAGS) (CHANGEDSINCE <cached-value> REPORTEXPUNGES)".

(Note, if <lastseenuid> is replaced with "*", this command will return flags for new messages as well)

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
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: A142 OK [READ-WRITE] SELECT completed
```


after that:

```
C: A143 UID FETCH 1:20 (FLAGS)
    (CHANGEDSINCE 20010715194032001 REPORTEXPUNGES)
```

```
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: A143 OK FETCH 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], 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-I02-EXPUNGED"
                       ;; [[Note to RFC Editor: fix before
                       ;; publication]]

message-data        =/ expunged-resp

expunged-resp       =  "VANISHED" [expunge-correlator] SP known-uids

expunge-correlator  =  SP "(" single-exp-correlator *(SP single-exp-
                       correlator) ")"
                       ;; Unless explicitly specified otherwise, ;;
                       all correlator types must be specified ;; only
                       once.

single-exp-correlator =  "TAG" SP tag-string ;; Correlator type
                       followed by parameters

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

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

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

expunged-param =      "REPORTEXPUNGES"
```

7. Security Considerations

It is believed that this extension doesn't raise any additional security concerns not already discussed in [RFC3501].

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.

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-I02-EXPUNGED [[anchor23: Note to RFC Editor: change before publication]] IMAP capability. IANA is requested to add it to the registry.

9. Acknowledgments

Thanks to Steve Hole, Cyrus Daboo, David Cridland and Michael Wener for encouraging me to write this document.

Valuable comments, both in agreement and in dissent, were received from David Cridland, Timo Sirainen, Michael Wener, Randall Gellens, Arnt Gulbrandsen, Peter Coates, Mark Crispin and Elwyn Davies.

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.

[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.

Author's Address

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

Email: Alexey.Melnikov@isode.com

Full Copyright Statement

Intellectual Property

http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

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