Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 22, 2007

D. Cridland
C. King
Isode Limited
June 20, 2007

Contexts for IMAP4
draft-cridland-imap-context-03

Status of this Memo

Copyright Notice

Abstract

The IMAP4rev1 protocol has powerful search facilities as part of the core protocol, but lacks the ability to create live, updated results which can be easily handled. This memo provides such an extension, and shows how it can be used to provide a facility similar to virtual mailboxes.

Table of Contents

1. Conventions used in this document

In examples, "C:" and "S:" indicate lines sent by the client messaging user agent and IMAP4rev1 ([IMAP]) server respectively. "//" indicates inline comments not part of the protocol exchange. Line breaks are liberally inserted for clarity. Examples are intended to be read in order, such that the state remains from one example to the next.

Although the examples show a server which supports [ESEARCH], this is not a strict requirement of this specification.

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 [KEYWORDS].

Other capitalised words are typically names of IMAP extensions or commands - these are uppercased for clarity only, and are case-insensitive.

[[Editorial comments are like this. XML2RFC working source is held at http://svn.dave.cridland.net/svn/ietf-drafts/draft-cridland-imap-context.xml]]

2. Introduction

Although the basic SEARCH command defined in [IMAP], and as enhanced by [ESEARCH], is relatively compact in its representation, this reduction saves only a certain amount of data, and huge mailboxes might overwhelm the storage available for results on even relatively high-end desktop machines.

The SORT command, defined in [SORT] provides useful features, but is hard to use effectively on changing mailboxes over low-bandwidth connections.

This memo borrows concepts from [ACAP], providing a windowed view onto search or sort results, as well as bandwidth and round-trip efficient updates, by providing two extensions, known as "ESORT" and "CONTEXT".

3. Extended Sort Syntax

Servers implementing the extended SORT provide a suite of extensions to the SORT and UID SORT commands defined in [SORT]. This allows for return options, as used with SEARCH and specified in [IMAP-ABNF], to be used with SORT in a similar manner.

The SORT and UID SORT commands are extended by the addition of an optional list of return options which follow a RETURN atom immediately after the command. If this is missing, the server will return results as specified in [SORT].

The extended SORT command always returns results in the requested sort order, but is otherwise identical in its behaviour to the extended SEARCH command defined in [IMAP-ABNF], as extended by [ESEARCH]. In particular, the extended SORT command returns results in an ESEARCH response.

3.1. ESORT extension

Servers advertising the capability "ESORT" support the return options specified in [ESEARCH], adapted as follows:

MIN
: Return the message number/UID of the lowest sorted message satisfying the search criteria.

MAX
: Return the message number/UID of the highest sorted message satisfying the search criteria.

ALL
: Return all message numbers/UIDs which match the search criteria, in the requested sort order, using a sequence-set. Note the use of ranges described below in Section 3.2.

COUNT
: As [ESEARCH].

3.2. Ranges in Extended Sort results

Any ranges given by the server, including those given as part of the sequence-set, in an ESEARCH response resulting from an extended SORT or UID SORT command MUST be ordered in increasing numerical order after expansion, as per usual [IMAP] rules.

In particular this means that 10:12 is equivalent to 12:10, and 10,11,12. To avoid confusion, servers SHOULD present ranges only

when the first seq-number is lower than the second; that is, either of the forms 10:12 or 10,11,12 is acceptable, but 12:10 SHOULD be avoided.

3.3. Extended SORT example

If the list of return options is present but empty, then the server provides the ALL return data item in an ESEARCH response. This is functionally equivalent to an unextended UID SORT command, but can use a smaller representation:

```
C: E01 UID SORT RETURN () (REVERSE DATE) UTF-8 UNDELETED
   UNKEYWORD $Junk
S: * ESEARCH (TAG "E01") UID ALL 23765,23764,23763,23761,[...]
S: E01 OK Sort completed
```

Note that the initial three results MUST NOT be represented as the range 23765:23763.

4. Contexts

4.1. Overview

This extension is present in any IMAP4rev1 server which includes the string "CONTEXT=SEARCH", and/or "CONTEXT=SORT", within its advertised capabilities.

In the case of CONTEXT=SEARCH, the server supports the extended SEARCH command syntax described in [IMAP-ABNF], and accepts three new return options.

Servers advertising CONTEXT=SORT also advertise the SORT capability, as described in [SORT], support the extended SORT command syntax described in Section 3, and accept in addition three new return options for this extended SORT.

These allow for notifications of changes to the results of SEARCH or SORT commands, and also allow for access to partial results.

A server advertising the CONTEXT=SEARCH extension will order all SEARCH results, whether from a UID SEARCH or SEARCH command, in mailbox order - that is, by message number and UID. Therefore, the UID SEARCH, SEARCH, UID SORT, or SORT command used - collectively known as the searching command - will always have an order, the requested order, which will be the mailbox order for UID SEARCH and SEARCH commands.

All of the return specifiers have no interaction with either each other or any return specifiers defined in [ESEARCH] or Section 3.1, however it is believed that implementations supporting CONTEXT will also support ESEARCH and ESORT.

4.2. Context Hint

The return option CONTEXT SHOULD be used by a client to indicate that subsequent use of the search criteria are likely. Servers MAY ignore this return option, or use it as a hint to maintain a full result cache, or index.

A client might choose to obtain a count of matching messages prior to obtaining actual results. Here, the client signals its intention to fetch the results themselves:

```
C: A01 SEARCH RETURN (CONTEXT COUNT) UNDELETED
   UNKEYWORD $Junk
S: * ESEARCH (TAG "A01") COUNT 23765
S: A01 OK Search completed.
```

4.3. Notifications of changes

The search return option UPDATE, if used by a client, causes the server to issue unsolicited notifications containing updates to the results which would be returned by an unmodified searching command. These update sets are carried in ADDTO and REMOVEFROM data items in ESEARCH/ESORT responses.

These ESEARCH responses carry a search correlator of the searching command, hence clients MUST NOT reuse tags, as already specified in Section 2.2.1 of [IMAP]. An attempt to use UPDATE where a tag is already in use with a previous searching command which itself used UPDATE SHALL result in the server rejecting the searching command with a BAD response.

Both ADDTO and REMOVEFROM data items SHOULD be delivered to clients in a timely manner, as and when results change, whether by new messages arriving in the mailbox, metadata such as flags being changed, or messages being expunged.

Typically, this would occur at the same time as the FETCH, EXISTS or EXPUNGE responses carrying the source of the change.

Updates will cease only when the mailbox is no longer selected, or when the CANCELUPDATE command, defined in Section 4.3.5, is issued by the client, whichever is sooner.

Unlike [ACAP], there is no requirement that a context need be created with CONTEXT to use UPDATE, and in addition, the lack of UPDATE with a CONTEXT does not affect the results caused by later searching commands - there is no snapshot facility.

There is no interaction between UPDATE and any other return options; therefore use of RETURN (UPDATE MIN), for example, does not notify about the minimum UID or sequence number, but notifies instead about all changes to the set of matching messages.

In particular, this means that a client using UPDATE and PARTIAL on the same search program MAY receive notifications about messages which do not currently interest it.

This time, the client will require notifications of updates, and chooses to obtain a count:

```
C: B01 UID SEARCH RETURN (UPDATE COUNT) DELETED
   KEYWORD $Junk
S: * ESEARCH (TAG "B01") COUNT 74
S: B01 OK Search completed, will notify.
// Note that the following is rejected, and has no effect:
C: B01 SORT RETURN (UPDATE) FLAGGED
S: B01 BAD Tag reuse
```

4.3.1. Refusing to update contexts

In some cases, the server MAY refuse to provide updates, such as if an internal limit on the number of update contexts is reached.

In this case, an untagged NO is generated during processing of the command with a response-code of NOUPDATE. The response-code contains, as argument, the tag of the search command for which the server is refusing to honour the UPDATE request.

Other return options specified will still be honoured.

Servers MUST provide at least one updating context per client, and SHOULD provide more - see Appendix B for strategies on reducing the impact of additional updating contexts. Since sorted contexts require a higher implementation cost than unsorted contexts, refusal to provide updates for a SORT command does not imply that SEARCH contexts will also be refused.

This time, the client will require notifications of updates, and chooses to obtain a count:

```
C: B02 UID SORT RETURN (UPDATE COUNT) $Junk
   KEYWORD $Junk
S: * ESEARCH (TAG "B02") COUNT 74
S: * NO [NOUPDATE "B02"] Too many contexts
S: B02 OK Search completed, will not notify.
```

Client handling might be to retry with a UID SEARCH command, or else cancel an existing context; see Section 4.3.5.

4.3.2. Common Features of ADDTO and REMOVEFROM

The result update set included in the return data item is specified as UIDs or message numbers, depending on how the UPDATE was specified. If the UPDATE was present in a SEARCH or SORT command, the results will be message numbers; in a UID SEARCH or UID SORT command, they will be UIDs.

The client MUST process ADDTO and REMOVEFROM return data items in order they appear, including those within a single ESEARCH response. Correspondingly, servers MUST generate ADDTO and REMOVEFROM responses such that the results are maintained in the requested order.

As with any response aside from EXPUNGE, ESEARCH responses carrying ADDTO and/or REMOVEFROM return data items MAY be sent at any time. In particular, servers MAY send such responses when no command is in progress, during the processing of any command, or when the client is using the IDLE facility described in [IDLE]. Implementors are recommended to read [NOTIFY] as a mechanism for clients to signal servers that they are willing to process responses at any time, and are also recommended to pay close attention to Section 5.3 of [IMAP].

It is expected that typical server implementations will emit ADDTO when they normally emit the causal FETCH or EXISTS, and similarly emit REMOVEFROM when they would normally emit the causal FETCH or EXPUNGE.

4.3.3. ADDTO Return Data Item

The ADDTO return data item contains, as payload, a list containing pairs of a position and a set of result updates in the requested order to be inserted at the position. Where the searching command is a SEARCH or UID SEARCH command, the position MAY be zero. Each pair is processed in the order that it appears.

If the position is non-zero, the result update is inserted at the

given position, meaning that the first result in the set will occupy the new position after insertion, and any prior existing result at that position will be shifted to later positions.

Where the position is zero, the client MAY insert the message numbers or UIDs in the result set such that the result set is maintained in mailbox order. In this case, servers are RECOMMENDED to order the result update into mailbox order to produce the shortest representation in set-syntax.

```
[...]
S: * 23762 FETCH (FLAGS (\Deleted \Seen))
S: * 23763 FETCH (FLAGS ($Junk \Seen))
S: * ESEARCH (TAG "B01") UID ADDTO (0 32768:32769)
```

Note that this example assumes messages 23762 and 23763 with UIDs 32768 and 32769 respectively previously had neither \Deleted nor $Junk set. Also note that only the ADDTO is included, and not the (now changed) COUNT.

If the searching command "C01" initially generated a result set of 4:5, then the following three responses are equivalent, and yield a result set of 1:5:

```
[...]
S: * ESEARCH (TAG "C01") UID ADDTO (1 3 1 2 1 1)
S: * ESEARCH (TAG "C01") UID ADDTO (1 3) ADDTO (1 1:2)
S: * ESEARCH (TAG "C01") UID ADDTO (1 1:3)
```

The last is the preferred representation.

4.3.4. REMOVEFROM Return Data Item

The REMOVEFROM return data item contains, as payload, a list containing pairs of a position and a set of result updates in the requested order to be removed starting from the position. Where the searching command is a SEARCH or UID SEARCH command, the position MAY be zero. Each pair is processed in the order that it appears.

If the position is non-zero, the results are removed at the given position, meaning that the first result in the set will occupy the given position before removal, and any prior existing result at that position will be shifted to earlier positions.

Where the position is zero, the client removes the message numbers or UIDs in the result set wherever they occur, and servers are RECOMMENDED to order the result set in mailbox order to obtain the

best benefit from the set-syntax.

Note that a REMOVEFROM containing message sequence numbers removed as a result of those messages being expunged MUST be sent prior to the expunge notification itself, in order that those sequence numbers remain valid.

Here, a message in the result set is expunged. The REMOVEFROM here is shown to happen without any command in progress, see Section 4.3.2. Note that EXPUNGE responses do not have this property.

```
[...]
S: * ESEARCH (TAG "B01") UID REMOVEFROM (0 32768)
C: B03 NOOP
S: * 23762 EXPUNGE
S: B03 OK Nothing done.
```

4.3.5. The CANCELUPDATE command

When a client no longer wishes to receive updates, it may issue the CANCELUPDATE command, which will prevent all updates to the contexts named in the arguments from being transmitted by the server. The command takes, as arguments, one or more tags of the commands used to request updates.

The server MAY free any resource associated with a context so disabled - however the client is free to issue further searching commands with the same criteria and requested order, including PARTIAL requests.

```
C: B04 CANCELUPDATE "B01"
S: B04 OK No further updates.
```

4.4. Partial results

The PARTIAL search return option causes the server to provide in an ESEARCH response a subset of the results denoted by the sequence range given as the mandatory argument. The first result is 1, thus the first 500 results would be obtained by a return option of "PARTIAL 1:500", and the second 500 by "PARTIAL 501:1000". This intentionally mirrors message sequence numbers.

Only a single PARTIAL search return option may be present in a single command.

For SEARCH results, the entire result set MUST be ordered in mailbox

order, that is, in UID or message sequence number order.

Where a PARTIAL search return option references results which do not exist, by using a range which starts or ends higher than the current number of results, then the server returns those results which are in the set. This yields a PARTIAL return data item which has, as payload, the original range and a potentially missing set of results which may be shorter than the extent of the range.

Clients need not request PARTIAL results in any particular order. Because mailboxes may change, clients will often wish to use PARTIAL in combination with UPDATE, especially if the intent is to walk a large set of results; however these return options do not interact - the UPDATE will provide notifications for all matching results.

```
// Recall from A01 that there are 23764 results.
C: A02 UID SEARCH RETURN (PARTIAL 23500:24000) UNDELETED
   UNKEYWORD $Junk
C: A03 UID SEARCH RETURN (PARTIAL 1:500) UNDELETED
   UNKEYWORD $Junk
C: A04 UID SEARCH RETURN (PARTIAL 24000:24500) UNDELETED
   UNKEYWORD $Junk
S: * ESEARCH (TAG "A02") UID PARTIAL (23500:24000 ...)
// 264 results in set syntax elided,
// this spans the end of the results.
S: A02 OK Completed.
S: * ESEARCH (TAG "A03") UID PARTIAL (1:500 ...)
// 500 results in set syntax elided.
S: A03 OK Completed.
S: * ESEARCH (TAG "A04") UID PARTIAL (24000:24500 NIL)
// No results are present, this is beyond the end of the results.
S: A04 OK Completed.
```

4.5. Caching results

Server implementations MAY cache results from a search or sort, whether or not hinted to by CONTEXT, in order to make subsequent searches more efficient, perhaps by recommencing a subsequent PARTIAL search where a previous search left off. However servers MUST behave identically whether or not internal caching is taking place, therefore any such cache is required to be updated as changes to the mailbox occur. An alternate strategy would be to discard results when any change occurs to the mailbox.

5. Formal Syntax

The collected formal syntax. This uses ABNF as defined in [ABNF]. It includes definitions from [IMAP], [IMAP-ABNF], and [SORT].

```
capability          =/ "CONTEXT=SEARCH" / "CONTEXT=SORT" / "ESORT"
    ;; <capability> from [IMAP]

command-select      =/ "CANCELUPDATE" 1*(SP quoted)
    ;; <command-select> from [IMAP]

addto-position      = number
    ;; Number may be 0 for SEARCH result additions.
    ;; <number> from [IMAP]

modifier-context    = "CONTEXT"

modifier-partial    = "PARTIAL" SP seq-range
    ;; <seq-range> from [IMAP]

modifier-update     = "UPDATE"

search-return-opt   =/ modifier-context / modifier-partial /
                       modifier-update
    ;; All conform to <search-return-opt>, from [IMAP-ABNF]

resp-text-code      =/ "NOUPDATE" SP quoted
    ;; <resp-text-code> from [IMAP]

ret-data-addto      = "ADDTO"
                       SP "(" addto-position SP sequence-set
                       *(SP addto-position SP sequence-set)
                       ")"
    ;; <sequence-set> from [IMAP]

ret-data-partial    = "PARTIAL"
                      SP "(" seq-range SP partial-results ")"
    ;; <seq-range> is the requested range.
    ;; <seq-range> from [IMAP]

partial-results     = sequence-set / "NIL"
    ;; <sequence-set> from [IMAP]
    ;; NIL indicates no results correspond to the requested range.
```

```
ret-data-removefrom = "REMOVEFROM"
                      SP "(" addto-position SP sequence-set
                      *(SP addto-position SP sequence-set)
                      ")"
    ;; <sequence-set> from [IMAP]

search-return-data  =/ ret-data-partial / ret-data-addto /
                       ret-data-removefrom
    ;; All conform to <search-return-data>, from [IMAP-ABNF]

sort                =/ extended-sort
    ;; <sort> from [SORT]

extended-sort       = ["UID" SP] "SORT" search-return-opts
                      SP sort-criteria SP search-criteria
    ;; <search-return-opts> from [IMAP-ABNF]
    ;; <sort-criteria> and <search-criteria> from [SORT]
```

6. Security Considerations

It is believed that this specification introduces no serious new security considerations. However, implementors are advised to refer to [IMAP].

Creation of contexts, both for UPDATE and PARTIAL, can benefit from storing potentially large result sets on the server. Implementors are advised to take care not to provide a method for denial of service (DoS) attacks based on this; the notes in Appendix B may aid in implementation decisions. Note that a server avoiding storing the results will have much increased I/O, which may also be an avenue for DoS attacks.

7. IANA Considerations

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

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

This document defines the ESORT, CONTEXT=SEARCH, and CONTEXT=SORT IMAP capabilities. IANA is requested to add them to the registry accordingly.

8. Acknowledgements

Much of the design of this extension can be found in ACAP. Valuable comments, both in agreement and in dissent, were received from Alexey Melnikov, Arnt Gulbrandsen, Cyrus Daboo, Filip Navara, Mark Crispin, Peter Coates, Philip Van Hoof, Randall Gellens, Timo Sirainen, Zoltan Ordogh and others, and many of these comments have had significant influence on the design or the text. The authors are grateful to all those involved, including those not mentioned here.

9. References

9.1. Normative References

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

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

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

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

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

[SORT] Crispin, M. and K. Murchison, "INTERNET MESSAGE ACCESS PROTOCOL - SORT AND THREAD EXTENSIONS", draft-ietf-imapext-sort-18 (work in progress), November 2006.

9.2. Informative References

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

[IDLE] Leiba, B., "IMAP4 IDLE command", RFC 2177, June 1997.

[NOTIFY] King, C., "The IMAP NOTIFY Extension", draft-gulbrandsen-imap-notify-06 (work in progress), May 2007.

Appendix A. Cookbook

A.1. Virtual Mailboxes

It is possible to use the facilities described within this memo to create a facility largely similar to a virtual mailbox, but handled on the client side.

Initially, the client SELECTs the real "backing" mailbox. Next, it can switch to a filtered view at any time by issuing a RETURN (COUNT UPDATE CONTEXT), and using RETURN (PARTIAL x:y) as the user scrolls, feeding the results into a FETCH as required to populate summary views.

A typically useful view is UID SORT (DATE) RETURN (...) UTF-8 UNSEEN UNDELETED, which can be used to show the mailbox sorted into INTERNALDATE order, filtered to only show messages which are unread and not yet deleted.

A.2. Trash Mailboxes

Certain contexts are particularly useful for client developers wishing to present something similar to the common trash mailbox metaphor in limited bandwidth. The simple criteria of UNDELETED only matches undeleted messages, and the corresponding DELETED search key can be used to display a per-mailbox trash-like virtual mailbox.

A.3. Immediate EXPUNGE notifications

The command "SEARCH RETURN (UPDATE) ALL" can be used to create a context which notifies immediately about expunged messages, yet will not affect message sequence numbers until the normal EXPUNGE message can be sent. This may be useful for clients wishing to show this behaviour without losing the benefit of sequence numbering.

A.4. Monitoring counts

A client need not maintain any result cache at all, but instead maintain a simple count of messages matching the search criteria. Typically, this would use the SEARCH command, as opposed to UID SEARCH, due to its smaller representation. Such usage might prove useful in monitoring the number of flagged, but unanswered, messages, for example, with "SEARCH RETURN (UPDATE COUNT) FLAGGED UNANSWERED".

A.5. Resynchronizing Contexts

The creation of a context, and immediate access to it, can all be accomplished in a single round-trip. Therefore, whilst it is

possible to elide resynchronization if no changes have occurred, it is simpler in most cases to resynchronize by simply recreating the context.

Appendix B. Server Implementation Notes

Although a server may cache the results, this is not mandated nor required, especially when the client uses SEARCH or UID SEARCH commands. UPDATE processing, for example, can be achieved efficiently by comparison of the old flag state (if any) and the new, and PARTIAL can be achieved by re-running the search until the suitable window is required. This is a result of there being no snapshot facility.

For example, on a new message, the server can simply test for matches against all current UPDATE context search programs, and for any that match, send the ADDTO return data.

Similarly, for a flag change on an existing message, the server can check whether the message matched with its old flags, whether it matches with new flags, and provide ADDTO or REMOVEFROM return data accordingly if these results differ.

For PARTIAL requests, the server can perform a full search, discarding results until the lower bound is hit, and stopping the search when sufficient results have been obtained.

With some additional state, it is possible to restart PARTIAL searches, thus avoiding performing the initial discard phase.

For the best performance, however, caching the full search results is needed, which can allow for faster responses at the expense of memory. One reasonable strategy would be to balance this trade-off at run-time, discarding search results after a suitable timeout, and regenerating them as required.

This yields state requirements of storing the search program for any UPDATE contexts, and optionally storing both search program and (updated) results for further contexts as required.

Note that in the absence of a server-side results cache, it may be impossible to know if an expunged message previously matched unless the original message is still available. Therefore some implementations may be forced into using a results cache in many circumstances.

UPDATE contexts created with SORT or UID SORT will almost certainly

require some form of results caching, however.

Authors' Addresses

Dave Cridland
Isode Limited
5 Castle Business Village
36, Station Road
Hampton, Middlesex TW12 2BX
GB

Email: dave.cridland@isode.com

Curtis King
Isode Limited
5 Castle Business Village
36, Station Road
Hampton, Middlesex TW12 2BX
GB

Email: cking@mumbo.ca

Full Copyright Statement

Intellectual Property

Acknowledgment

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