INTERNET MESSAGE ACCESS PROTOCOL - THREAD EXTENSION

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC 2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

To view the list Internet-Draft Shadow Directories, see http://www.ietf.org/shadow.html.

A revised version of this draft document will be submitted to the RFC editor as a Proposed Standard for the Internet Community. Discussion and suggestions for improvement are requested, and should be sent to ietf-imapext@IMC.ORG. This document will expire before 21 August 2002. Distribution of this memo is unlimited.

Abstract

This document describes the server-based threading extension to the IMAP4rev1 protocol. This extension provides substantial performance improvements for IMAP clients which offer threaded views.

A server which supports this extension indicates this with one or more capability names consisting of "THREAD=" followed by a supported threading algorithm name as described in this document. This provides for future upwards-compatible extensions.

Extracted Subject Text

Threading uses a version of the subject which has specific subject artifacts of deployed Internet mail software removed. Due to the complexity of these artifacts, the formal syntax for the subject extraction rules is ambiguous. The following procedure is followed to determine the actual "base subject" which is used to thread:

(1) Convert any RFC 2047 encoded-words in the subject to UTF-8. Convert all tabs and continuations to space. Convert all multiple spaces to a single space.

(2) Remove all trailing text of the subject that matches the subj-trailer ABNF, repeat until no more matches are possible.

(3) Remove all prefix text of the subject that matches the subj-leader ABNF.

(4) If there is prefix text of the subject that matches the subj-blob ABNF, and removing that prefix leaves a non-empty subj-base, then remove the prefix text.

(5) Repeat (3) and (4) until no matches remain.

Note: it is possible to defer step (2) until step (6), but this requires checking for subj-trailer in step (4).

(6) If the resulting text begins with the subj-fwd-hdr ABNF and ends with the subj-fwd-trl ABNF, remove the subj-fwd-hdr and subj-fwd-trl and repeat from step (2).

(7) The resulting text is the "base subject" used in threading.

All servers and disconnected clients MUST use exactly this algorithm when threading. Otherwise there is potential for a user to get inconsistent results based on whether they are running in connected or disconnected IMAP mode.

Sent Date

As used in this document, the term "sent date" refers to the date and time from the Date: header, adjusted by time zone. This differs from date-related criteria in SEARCH, which use just the date and not the time, nor adjusts by time zone.

Additional Commands

This command is an extension to the IMAP4rev1 base protocol.

The section header is intended to correspond with where it would be located in the main document if it was part of the base specification.

6.3.THREAD. THREAD Command

Arguments: threading algorithm
charset specification
searching criteria (one or more)

Data: untagged responses: THREAD

Result: OK - thread completed
NO - thread error: can't thread that charset or criteria
BAD - command unknown or arguments invalid

The THREAD command is a variant of SEARCH with threading semantics for the results. Thread has two arguments before the searching criteria argument; a threading algorithm, and the searching charset. Note that unlike SEARCH, the searching charset argument is mandatory.

There is also a UID THREAD command which corresponds to THREAD the way that UID SEARCH corresponds to SEARCH.

The THREAD command first searches the mailbox for messages that match the given searching criteria using the charset argument for the interpretation of strings in the searching criteria. It then returns the matching messages in an untagged THREAD response, threaded according to the specified threading algorithm.

Sorting is in ascending order. Earlier dates sort before later dates; smaller sizes sort before larger sizes; and strings are sorted according to ascending values established by their collation algorithm (see under "Internationalization Considerations").

The defined threading algorithms are as follows:

ORDEREDSUBJECT

The ORDEREDSUBJECT threading algorithm is also referred to as "poor man's threading." The searched messages are sorted by subject and then by the sent date. The messages are then split into separate threads, with each thread containing messages with the same extracted subject text. Finally, the threads are sorted by the sent date of the first message in the thread.

Note that each message in a thread is a child (as opposed to a sibling) of the previous message.

REFERENCES

The REFERENCES threading algorithm is based on the algorithm written by Jamie Zawinski which was used in "Netscape Mail and News" versions 2.0 through 3.0. For details, see http://www.jwz.org/doc/threading.html.

This algorithm threads the searched messages by grouping them together in parent/child relationships based on which messages are replies to others. The parent/child relationships are built using two methods: reconstructing a message's ancestry using the references contained within it; and checking the subject of a message to see if it is a reply to (or forward of) another.

Note: "Message ID" in the following description refers to a normalized form of the msg-id in [RFC 2822]. The actual text in an RFC 2822 may use quoting, resulting in multiple ways of expressing the same Message ID. Implementations of the REFERENCES threading algorithm MUST normalize any msg-id in order to avoid false non-matches due to differences in quoting.

For example, the msg-id
<"01KF8JCEOCBS0045PS"@xxx.yyy.com>
and the msg-id
<01KF8JCEOCBS0045PS@xxx.yyy.com>
MUST be interpreted as being the same Message ID.

The references used for reconstructing a message's ancestry are found using the following rules:

If a message contains a [NEWS]-style References header line, then use the Message IDs in the References header line as the references.

If a message does not contain a References header line, or the References header line does not contain any valid Message IDs, then use the first (if any) valid Message ID found in the In-Reply-To header line as the only reference (parent) for this message.

Note: Although [RFC 2822] permits multiple Message IDs in the In-Reply-To header, in actual practice this discipline has not been followed. For example, In-Reply-To headers have been observed with email addresses after the Message ID, and there are no good heuristics for software to determine the difference. This is not a problem with the References header however.

If a message does not contain an In-Reply-To header line, or the In-Reply-To header line does not contain a valid Message ID, then the message does not have any references (NIL).

The REFERENCES algorithm is significantly more complex than ORDEREDSUBJECT and consists of six main steps. These steps are outlined in detail below.

(1) For each searched message:

(A) Using the Message IDs in the message's references, link the corresponding messages (those whose Message-ID header line contains the given reference Message ID) together as parent/child. Make the first reference the parent of the second (and the second a child of the first), the second the parent of the third (and the third a child of the second), etc. The following rules govern the creation of these links:

If a message does not contain a Message-ID header line, or the Message-ID header line does not contain a valid Message ID, then assign a unique Message ID to this message.

If two or more messages have the same Message ID, assign a unique Message ID to each of the duplicates.

If no message can be found with a given Message ID, create a dummy message with this ID. Use this dummy message for all subsequent references to this ID.

If a message already has a parent, don't change the existing link. This is done because the References header line may have been truncated by a MUA. As a

result, there is no guarantee that the messages corresponding to adjacent Message IDs in the References header line are parent and child.

Do not create a parent/child link if creating that link would introduce a loop. For example, before making message A the parent of B, make sure that A is not a descendent of B.

Note: Message ID comparisons are case-sensitive.

(B) Create a parent/child link between the last reference (or NIL if there are no references) and the current message. If the current message already has a parent, it is probably the result of a truncated References header line, so break the current parent/child link before creating the new correct one. As in step 1.A, do not create the parent/child link if creating that link would introduce a loop. Note that if this message has no references, that it will now have no parent.

Note: The parent/child links created in steps 1.A and 1.B MUST be kept consistent with one another at ALL times.

(2) Gather together all of the messages that have no parents and make them all children (siblings of one another) of a dummy parent (the "root"). These messages constitute the first (head) message of the threads created thus far.

(3) Prune dummy messages from the thread tree. Traverse each thread under the root, and for each message:

If it is a dummy message with NO children, delete it.

If it is a dummy message with children, delete it, but promote its children to the current level. In other words, splice them in with the dummy's siblings.

Do not promote the children if doing so would make them children of the root, unless there is only one child.

(4) Sort the messages under the root (top-level siblings only) by sent date. In the case of an exact match on sent date or if either of the Date: headers used in a comparison can not be parsed, use the order in which the messages appear in the mailbox (that is, by sequence number) to determine the order. In the case of a dummy message, sort its children by sent date and then use the first child for the top-level sort.

(5) Gather together messages under the root that have the same extracted subject text.

(A) Create a table for associating extracted subjects with messages.

(B) Populate the subject table with one message per extracted subject. For each child of the root:

(i) Find the subject of this thread by extracting the base subject from the current message, or its first child if the current message is a dummy.

(ii) If the extracted subject is empty, skip this message.

(iii) Lookup the message associated with this extracted subject in the table.

(iv) If there is no message in the table with this subject, add the current message and the extracted subject to the subject table.

Otherwise, replace the message in the table with the current message if the message in the table is not a dummy AND either of the following criteria are true:

The current message is a dummy, OR

The message in the table is a reply or forward (its original subject contains a subj-refwd part and/or a "(fwd)" subj-trailer) and the current message is not.

(C) Merge threads with the same subject. For each child of the root:

(i) Find the subject of this thread as in step 4.B.i above.

(ii) If the extracted subject is empty, skip this message.

(iii) Lookup the message associated with this extracted subject in the table.

(iv) If the message in the table is the current message, skip this message.

Otherwise, merge the current message with the one in the table using the following rules:

If both messages are dummies, append the current message's children to the children of the message in the table (the children of both messages become siblings), and then delete the current message.

If the message in the table is a dummy and the current message is not, make the current message a child of the message in the table (a sibling of it's children).

If the current message is a reply or forward and the message in the table is not, make the current message a child of the message in the table (a sibling of it's children).

Otherwise, create a new dummy message and make both the current message and the message in the table children of the dummy. Then replace the message in the table with the dummy message.

Note: Subject comparisons are case-insensitive, as described under "Internationalization Considerations."

(6) Traverse the messages under the root and sort each set of siblings by sent date. Traverse the messages in such a way that the "youngest" set of siblings are sorted first, and the "oldest" set of siblings are sorted last (grandchildren are sorted before children, etc). In the case of an exact match on sent date or if either of the Date: headers used in a comparison can not be parsed, use the order in which the messages appear in the mailbox (that is, by sequence number) to determine the order. In the case of a dummy message (which can only occur with top-level siblings), use its first child for sorting.

Example:
```
C: A283 THREAD ORDEREDSUBJECT UTF-8 SINCE 5-MAR-2000
S: * THREAD (166)(167)(168)(169)(172)(170)(171)
   (173)(174 175 176 178 181 180)(179)(177 183
    182 188 184 185 186 187 189)(190)(191)(192)
   (193)(194 195)(196 197 198)(199)(200 202)(201)
   (203)(204)(205)(206 207)(208)
S: A283 OK THREAD completed
C: A284 THREAD ORDEREDSUBJECT US-ASCII TEXT "gewp"
S: * THREAD
```

```
S: A284 OK THREAD completed
C: A285 THREAD REFERENCES UTF-8 SINCE 5-MAR-2000
S: * THREAD (166)(167)(168)(169)(172)((170)(179))
   (171)(173)((174)(175)(176)(178)(181)(180))
   ((177)(183)(182)(188 (184)(189))(185 186)(187))
   (190)(191)(192)(193)((194)(195 196))(197 198)
   (199)(200 202)(201)(203)(204)(205 206 207)(208)
S: A285 OK THREAD completed
```

Note: The line breaks in the first and third client responses are for editorial clarity and do not appear in real THREAD responses.

Additional Responses

This response is an extension to the IMAP4rev1 base protocol.

The section heading of this response is intended to correspond with where it would be located in the main document.

7.2.THREAD. THREAD Response

Data: zero or more threads

The THREAD response occurs as a result of a THREAD or UID THREAD command. It contains zero or more threads. A thread consists of a parenthesized list of thread members.

Thread members consist of zero or more message numbers, delimited by spaces, indicating successive parent and child. This continues until the thread splits into multiple sub-threads, at which point the thread nests into multiple sub-threads with the first member of each subthread being siblings at this level. There is no limit to the nesting of threads.

The messages numbers refer to those messages that match the search criteria. For THREAD, these are message sequence numbers; for UID THREAD, these are unique identifiers.

Example: S: * THREAD (2)(3 6 (4 23)(44 7 96))

The first thread consists only of message 2. The second thread consists of the messages 3 (parent) and 6 (child), after which it splits into two subthreads; the first of which contains messages 4 (child of 6, sibling of 44) and 23 (child of 4), and the second of which contains messages 44 (child of 6, sibling of 4), 7 (child of 44), and 96 (child of 7). Since some later messages are parents of earlier messages, the messages were probably moved from some other mailbox at different times.

```
-- 2

-- 3
   \-- 6
       |-- 4
       |   \-- 23
       |
       \-- 44
            \-- 7
                \-- 96
```

Example: S: * THREAD ((3)(5))

In this example, 3 and 5 are siblings of a parent which does not match the search criteria (and/or does not exist in the mailbox); however they are members of the same thread.

Formal Syntax of THREAD commands and Responses

```
   thread-data       = "THREAD" [SP 1*thread-list]

   thread-list       = "(" thread-members / thread-nested ")"

   thread-members    = nz-number *(SP nz-number) [SP thread-nested]

   thread-nested     = 2*thread-list

   thread            = ["UID" SP] "THREAD" SP thread-algorithm
                       SP search-charset 1*(SP search-key)

   thread-algorithm  = "ORDEREDSUBJECT" / "REFERENCES" / atom
```

The following syntax describes subject extraction rules (2)-(6):

```
   subject         = *subj-leader [subj-middle] *subj-trailer

   subj-refwd      = ("re" / ("fw" ["d"])) *WSP [subj-blob] ":"

   subj-blob       = "[" *BLOBCHAR "]" *WSP

   subj-fwd        = subj-fwd-hdr subject subj-fwd-trl

   subj-fwd-hdr    = "[fwd:"

   subj-fwd-trl    = "]"

   subj-leader     = (*subj-blob subj-refwd) / WSP

   subj-middle     = *subj-blob (subj-base / subj-fwd)
                   ; last subj-blob is subj-base if subj-base would
                   ; otherwise be empty

   subj-trailer    = "(fwd)" / WSP

   subj-base       = NONWSP *([*WSP] NONWSP)
                   ; can be a subj-blob

   BLOBCHAR        = %x01-5a / %x5c / %x5e-7f
                   ; any CHAR except '[' and ']'

   NONWSP          = %x01-08 / %x0a-1f / %x21-7f
                   ; any CHAR other than WSP
```

Security Considerations

Security issues are not discussed in this memo.

Internationalization Considerations

By default, strings are threaded according to the "minimum sorting collation algorithm". All implementations of THREAD MUST implement the minimum sorting collation algorithm.

In the minimum sorting collation algorithm, the Basic Latin alphabetics (U+0041 to U+005A uppercase, U+0061 to U+007A lowercase) are sorted in a case-insensitive fashion; that is, "A" (U+0041) and "a" (U+0061) are treated as exact equals. The characters U+005B to U+0060 are sorted after the Basic Latin alphabetics; for example, U+005E is sorted after U+005A and U+007A. All other characters are sorted according to their octet values, as expressed in UTF-8. No attempt is made to treat composed characters specially, or to do case-insensitive comparisons of composed characters.

> Note: this means, among other things, that the composed characters in the Latin-1 Supplement are not compared in what would be considered an ISO 8859-1 "case-insensitive" fashion. Case comparison rules for characters with diacriticals differ between languages; the minimum sorting collation does not attempt to deal with this at all. This is reserved for other sorting collations, which may be language-specific.

```
;;;   *** ITEM FOR DISCUSSION ***
;;; THERE IS SOME CONCERN THAT THIS MINIMUM COLLATION IS TOO MINIMAL,
;;; AND THAT THE "GENERIC UNICODE SORTING COLLATION" DISCUSSED BELOW
;;; NEEDS TO BE THE MINIMUM.  ONE SUGGESTION IS UNICODE TECHNICAL
;;; STANDARD 10 (TR-10).  IF THIS IS THE MINIMUM, THAT REQUIRES THAT
;;; ALL IMPLEMENTATIONS OF SORT AND THREAD BE UNICODE-SAVVY AT LEAST
;;; TO THE POINT OF IMPLEMENTATION TR-10.  IS THIS REALISTIC?  DOES
;;; THIS RAISE EXCESSIVE IMPLEMENTATION BARRIERS?
```

Other sorting collations, and the ability to change the sorting collation, will be defined in a separate document dealing with IMAP internationalization.

It is anticipated that there will be a generic Unicode sorting collation, which will provide generic case-insensitivity for alphabetic scripts, specification of composed character handling, and language-specific sorting collations. A server which implements non-default sorting collations will modify its sorting behavior according to the selected sorting collation.

Non-English translations of "Re" or "Fw"/"Fwd" are not specified for removal in the extracted subject text process. By specifying that only the English forms of the prefixes are used, it becomes a simple display time task to localize the prefix language for the user. If, on the other hand, prefixes in multiple languages are permitted, the result is a geometrically complex, and ultimately unimplementable, task. In order to improve the ability to support non-English display in Internet mail clients, only the English form of these prefixes should be transmitted in Internet mail messages.

A. References

[ABNF] Crocker, D., and Overell, P. "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[NEWS] Horton, M., and Adams, R., "Standard for interchange of USENET messages", RFC-1036, AT&T Bell Laboratories and Center for Seismic Studies, December, 1987.

[RFC-2822] Resnick, P. "Internet Message Format", RFC 2822, April 2001.

Author's Address

Mark R. Crispin
Networks and Distributed Computing
University of Washington
4545 15th Avenue NE
Seattle, WA 98105-4527

Phone: (206) 543-5762

EMail: MRC@CAC.Washington.EDU

Kenneth Murchison
Oceana Matrix Ltd.
21 Princeton Place
Orchard Park, NY 14127

Phone: (716) 662-8973 x26

EMail: ken@oceana.com