INTERNET MESSAGE ACCESS PROTOCOL - SORT AND THREAD EXTENSIONS

Status of this Memo

Abstract

   This document describes the base-level server-based sorting and
   threading extensions to the [IMAP] protocol. These extensions
   provide substantial performance improvements for IMAP clients which
   offer sorted and threaded views.

1. Introduction

   The SORT and THREAD extensions to the [IMAP] protocol provide a means

of server-based sorting and threading of messages, without requiring
that the client download the necessary data to do so itself. This is
particularly useful for online clients as described in [IMAP-MODELS].

A server which supports the base-level SORT extension indicates this
with a capability name which starts with "SORT". Future,
upwards-compatible extensions to the SORT extension will all start
with "SORT", indicating support for this base level.

A server which supports the THREAD 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.

A server which implements the SORT and/or THREAD extensions SHOULD
also implement the COMPARATOR extension as described in [IMAP-I18N].

2. Terminology

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].

The word "can" (not "may") is used to refer to a possible
circumstance or situation, as opposed to an optional facility of the
protocol.

"User" is used to refer to a human user, whereas "client" refers to
the software being run by the user.

In examples, "C:" and "S:" indicate lines sent by the client and
server respectively.

2.1 Base Subject

Subject sorting and threading use the "base subject," which has
specific subject artifacts 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
"base subject", using the [ABNF] formal syntax rules described in
section 5:

    (1) Convert any RFC 2047 encoded-words in the subject to
    UTF-8 as described in "internationalization
    considerations." 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 the
SORT.

All servers and disconnected (as described in [IMAP-MODELS]) clients
MUST use exactly this algorithm to determine the "base subject".
Otherwise there is potential for a user to get inconsistent results
based on whether they are running in connected or disconnected mode.

2.2 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 to normalize to
UTC. For example, "31 Dec 2000 16:01:33 -0800" is equivalent to the
UTC date and time of "1 Jan 2001 00:01:33 +0000".

If the time zone is invalid, the date and time SHOULD be treated as
UTC. If the time is also invalid, the time SHOULD be treated as
00:00:00. If there is no valid date or time, the date and time
SHOULD be treated as 00:00:00 on the earliest possible date.

This differs from the date-related criteria in the SEARCH command
(described in [IMAP] section 6.4.4), which use just the date and not
the time, and are not adjusted by time zone.

3. Additional Commands

These commands are extension to the [IMAP] base protocol.

The section headings are intended to correspond with where they would
be located in the main document if they were part of the base
specification.

BASE.6.4.SORT. SORT Command

```
Arguments:   sort program
             charset specification
             searching criteria (one or more)

Data:        untagged responses: SORT

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

The SORT command is a variant of SEARCH with sorting semantics for
the results. Sort has two arguments before the searching criteria
argument; a parenthesized list of sort criteria, and the searching
charset.

The charset argument is mandatory (unlike SEARCH) and indicates
the [CHARSET] of the strings that appear in the searching
criteria. The US-ASCII and UTF-8 charsets MUST be implemented.
All other charsets are optional.

There is also a UID SORT command which returns unique identifiers
instead of message sequence numbers. Note that there are separate
searching criteria for message sequence numbers and UIDs; thus the
arguments to UID SORT are interpreted the same as in SORT. This
is analogous to the behavior of UID SEARCH, as opposed to UID
COPY, UID FETCH, or UID STORE.

The SORT 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 SORT response, sorted
according to one or more sort criteria.

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").

If two or more messages exactly match according to the sorting
criteria, these messages are sorted according to the order in
which they appear in the mailbox. In other words, there is an
implicit sort criterion of "sequence number".

When multiple sort criteria are specified, the result is sorted in
the priority order that the criteria appear. For example,
(SUBJECT DATE) will sort messages in order by their base subject
text; and for messages with the same base subject text will sort
by their sent date.

Untagged EXPUNGE responses are not permitted while the server is responding to a SORT command, but are permitted during a UID SORT command.

The defined sort criteria are as follows. Refer to the Formal Syntax section for the precise syntactic definitions of the arguments. If the associated [RFC-822](#) header for a particular criterion is absent, it is treated as the empty string. The empty string always collates before non-empty strings.

ARRIVAL
   Internal date and time of the message. This differs from the ON criteria in SEARCH, which uses just the internal date.

CC
   [[IMAP](#)] addr-mailbox of the first "cc" address.

DATE
   Sent date and time from the Date: header, adjusted by time zone. This differs from the SENTON criteria in SEARCH, which uses just the date and not the time, nor adjusts by time zone. If the sent date can not be determined (a Date: header is missing or can not be parsed), the INTERNALDATE for that message is used as the sent date.

FROM
   [[IMAP](#)] addr-mailbox of the first "From" address.

REVERSE
   Followed by another sort criterion, has the effect of that criterion but in reverse (descending) order.
      Note: REVERSE only reverses a single criterion, and does not affect the implicit "sequence number" sort criterion if all other criteria are identicial. Consequently, a sort of REVERSE SUBJECT is not the same as a reverse ordering of a SUBJECT sort. This can be avoided by use of additional criteria, e.g. SUBJECT DATE vs. REVERSE SUBJECT REVERSE DATE. In general, however, it's better (and faster, if the client has a "reverse current ordering" command) to reverse the results in the client instead of issuing a new SORT.

SIZE
   Size of the message in octets.

SUBJECT
   Base subject text.

TO
   [[IMAP](#)] addr-mailbox of the first "To" address.

Example: C: A282 SORT (SUBJECT) UTF-8 SINCE 1-Feb-1994

```
            S: * SORT 2 84 882
            S: A282 OK SORT completed
            C: A283 SORT (SUBJECT REVERSE DATE) UTF-8 ALL
            S: * SORT 5 3 4 1 2
            S: A283 OK SORT completed
            C: A284 SORT (SUBJECT) US-ASCII TEXT "not in mailbox"
            S: * SORT
            S: A284 OK SORT completed
```

BASE.6.4.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.

     The charset argument is mandatory (unlike SEARCH) and indicates
     the [CHARSET] of the strings that appear in the searching
     criteria. The US-ASCII and UTF-8 charsets MUST be implemented.
     All other charsets are optional.

     There is also a UID THREAD command which returns unique
     identifiers instead of message sequence numbers. Note that there
     are separate searching criteria for message sequence numbers and
     UIDs; thus the arguments to UID THREAD are interpreted the same as
     in THREAD. This is analogous to the behavior of UID SEARCH, as
     opposed to UID COPY, UID FETCH, or UID STORE.

     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.

     All collation is in ascending order. Earlier dates collate before
     later dates and strings are collated according to ascending values
     established by their collation algorithm (see under
     "Internationalization Considerations").

     Untagged EXPUNGE responses are not permitted while the server is

responding to a THREAD command, but are permitted during a UID
THREAD command.

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
   base subject and then by the sent date. The messages are then
   split into separate threads, with each thread containing
   messages with the same base subject text. Finally, the threads
   are sorted by the sent date of the first message in the thread.

   The first message of each thread are siblings of each other
   (the "root"). The second message of a thread is the child of
   the first message, and subsequent messages of the thread are
   siblings of the second message and hence children of the
   message at the root. Hence, there are no grandchildren in
   ORDEREDSUBJECT threading.

      Note: early drafts of this specification specified
      that each message in an ORDEREDSUBJECT thread is a child
      (as opposed to a sibling) of the previous message. This
      is now deprecated. For compatibility with servers which
      may still use the old definition, client implementations
      SHOULD treat descendents of a child as being siblings of
      that child.

      This is because the old definition mistakenly indicated
      that there was a parent/child relationship between
      successive messages in a thread; when in fact there was
      only a chronological relationship. In clients which
      indicate parent/child relationships in a thread tree,
      this would indicate levels of descent which did not
      exist.

REFERENCES

   The REFERENCES threading algorithm is based on the [THREADING]
   algorithm written used in "Netscape Mail and News" versions 2.0
   through 3.0. 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 original (not base) subject of a message to
   see if it is a reply to (or forward of) another message.

      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 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 message
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).

A message is considered to be a reply or forward if the base
subject extraction rules, applied to the original subject,
remove any of the following: a subj-refwd, a "(fwd)"
subj-trailer, or a subj-fwd-hdr and subj-fwd-trl.

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, then
    only use that Message ID in the first (lowest sequence
    number) message, and assign a unique Message ID to each
    of the subsequent messages with a duplicate of that
    Message ID.

    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, 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. If the sent date can not
be determined (a Date: header is missing or can not be parsed),
the INTERNALDATE for that message is used as the sent date.

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

(A) Create a table for associating base subjects with
messages, called the subject table.

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

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

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

(iii) Look up the message associated with the thread
subject in the subject table.

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

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

The current message is a dummy, OR

The message in the subject table is a reply or forward
and the current message is not.

then replace the message in the subject table with the

current message.

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

   (i) Find the message's thread subject as in step 5.B.i
   above.

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

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

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

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

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

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

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

      Otherwise, create a new dummy message and make both
      the current message and the message in the subject
      table children of the dummy. Then replace the message
      in the subject 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 server
        responses are for editorial clarity and do not appear in
        real THREAD responses.

[4](#). **Additional Responses**

    These responses are extensions to the [[IMAP](#)] base protocol.

    The section headings of these responses are intended to correspond
    with where they would be located in the main document.

BASE.7.2.SORT. SORT Response

    Data: zero or more numbers

        The SORT response occurs as a result of a SORT or UID SORT
        command. The number(s) refer to those messages that match the
        search criteria. For SORT, these are message sequence numbers;
        for UID SORT, these are unique identifiers. Each number is
        delimited by a space.

    Example: S: * SORT 2 3 6

BASE.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.

5. Formal Syntax of SORT and THREAD Commands and Responses

The following syntax specification uses the Augmented Backus-Naur
Form (ABNF) notation as specified in [ABNF]. It also uses [ABNF]
rules defined in [IMAP].

sort   = ["UID" SP] "SORT" SP sort-criteria SP search-criteria

```
sort-criteria   = "(" sort-criterion *(SP sort-criterion) ")"

sort-criterion  = ["REVERSE" SP] sort-key

sort-key        = "ARRIVAL" / "CC" / "DATE" / "FROM" / "SIZE" /
                   "SUBJECT" / "TO"

thread          = ["UID" SP] "THREAD" SP thread-alg SP search-criteria

thread-alg      = "ORDEREDSUBJECT" / "REFERENCES" / thread-alg-ext

thread-alg-ext  = atom
                     ; New algorithms MUST be registered with IANA

search-criteria = charset 1*(SP search-key)

charset         = atom / quoted
                     ; CHARSET values MUST be registered with IANA

sort-data       = "SORT" *(SP nz-number)

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


    The following syntax describes base 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-ff
                          ; any CHAR8 except '[' and ']'

NONWSP          = %x01-08 / %x0a-1f / %x21-ff
                          ; any CHAR8 other than WSP
```

[6](#). Security Considerations

The SORT and THREAD extensions do not raise any security
considerations that are not present in the base [IMAP] protocol, and
these issues are discussed in [IMAP]. Nevertheless, it is important
to remember that [IMAP] protocol transactions, including message
data, are sent in the clear over the network unless protection from
snooping is negotiated, either by the use of STARTTLS, privacy
protection is negotiated in the AUTHENTICATE command, or some other
protection mechanism is in effect.

[7](#). Internationalization Considerations

As stated in the introduction, the server SHOULD support the
[IMAP-I18N] COMPARATOR extension and follow its rules to perform
collations in the SORT and THREAD extensions.

If the server does not support COMPARATOR, strings MUST be collated
according to the i;unicode-casemap collation described in
[UNICASEMAP].

As described in [IMAP-I18N], strings in charsets other than US-ASCII
and UTF-8 MUST be converted to UTF-8 and compared in ascending order
according to the selected or active collation algorithm. If the
server does not support the [IMAP-I18N] COMPARATOR extension, the
collation algorithm used is the "en;ascii-casemap" collation
described in [COMPARATOR].

Translations of the "re" or "fw"/"fwd" tokens are not specified for
removal in the base subject extraction process. An attempt to add
such translated tokens would result in a geometrically complex, and
ultimately unimplementable, task.

Instead, note that [RFC-2822] section 3.6.5 recommends that "re:"
(from the Latin "res", in the matter of) be used to identify a reply.
Although it is evident that, from the multiple forms of token to
identify a forwarded message, there is considerable variation found
in the wild, the variations are (still) manageable. Consequently, it
is suggested that "re:" and one of the variations of the tokens for
forward supported by the base subject extraction rules be adopted for
Internet mail messages, since doing so makes it a simple display time
task to localize the token language for the user.

8. IANA Considerations

   [IMAP] capabilities are registered by publishing a standards track or
   IESG approved experimental RFC. This document constitutes
   registration of the SORT and THREAD capabilities in the [IMAP]
   capabilities registry.

   This document creates a new [IMAP] threading algorithms registry,
   which registers threading algorithms by publishing a standards track
   or IESG approved experimental RFC. This document constitutes
   registration of the ORDEREDSUBJECT and REFERENCES algorithms in that
   registry.

9. Normative References

   The following documents are normative to this document:

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

   [CHARSET]   Freed, N. and Postel, J. "IANA Character Set
                       Registration Procedures", RFC 2978, October
                       2000.

   [COMPARATOR]   Newman, C. and Duerst, M. "Internet
                       Appplication Protocol Collation Registry",
                       RFC 4790, March 2007

   [IMAP]   Crispin, M. "Internet Message Access Protocol -
                       Version 4rev1", RFC 3501, March 2003.

   [IMAP-I18N]   Newman, C. and Gulbrandsen, A. "Internet
                       Message Access Protocol Internationalization",
                       Work in Progress.

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

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

   [UNICASEMAP]   Crispin, M. "i;unicode-casemap - Simple Unicode
                       Collation Algorithm", Work in Progress.

10. Informative References

The following documents are informative to this document:

[IMAP-MODELS]   Crispin, M. "Distributed Electronic Mail Models
                     in IMAP4", RFC 1733, December 1994.

[THREADING]   Zawinski, J. "Message Threading",
                     http://www.jwz.org/doc/threading.html,
                     1997-2002.

Appendices

Author's Address

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

    Phone: +1 (206) 543-5762

    EMail: MRC@CAC.Washington.EDU

    Kenneth Murchison
    Carnegie Mellon University
    5000 Forbes Avenue
    Cyert Hall 285
    Pittsburgh, PA 15213

    Phone: +1 (412) 268-2638
    Email: murch@andrew.cmu.edu

Acknowledgement