Network Working Group
Request for Comments: 4466
Updates: 2088, 2342, 3501, 3502, 3516
Category: Standards Track

A. Melnikov
Isode Ltd.
C. Daboo

April 2006

Collected Extensions to IMAP4 ABNF

Status of This Memo

Copyright Notice

Abstract

Over the years, many documents from IMAPEXT and LEMONADE working groups, as well as many individual documents, have added syntactic extensions to many base IMAP commands described in RFC 3501. For ease of reference, this document collects most of such ABNF changes in one place.

This document also suggests a set of standard patterns for adding options and extensions to several existing IMAP commands defined in RFC 3501. The patterns provide for compatibility between existing and future extensions.

This document updates ABNF in RFCs 2088, 2342, 3501, 3502, and 3516. It also includes part of the errata to RFC 3501. This document doesn’t specify any semantic changes to the listed RFCs.

Table of Contents

1. Introduction

1.1. Purpose of This Document

This document serves several purposes:

1. rationalize and generalize ABNF for some existing IMAP extensions;
2. collect the ABNF in one place in order to minimize cross references between documents;
3. define building blocks for future extensions so that they can be used together in a compatible way.

It is expected that a future revision of this document will be incorporated into a revision of RFC 3501.

This document updates ABNF in RFCs 2088, 2342, 3501, 3502, and 3516. It also includes part of the errata to RFC 3501. This document doesn't specify any semantic changes to the listed RFCs.

The ABNF in section 6 of RFC 2342 got rewritten to conform to the ABNF syntax as defined in RFC 4234 and to reference new non-terminals from RFC 3501. It was also restructured to allow for better readability. There were no changes "on the wire".

Section 2 extends ABNF for SELECT, EXAMINE, CREATE, RENAME, FETCH/UID FETCH, STORE/UID STORE, SEARCH, and APPEND commands in a consistent manner. Extensions to all the commands but APPEND have the same

structure. Extensibility for the APPEND command was done slightly differently in order to preserve backward compatibility with existing extensions.

Section 2 also defines a new ESEARCH response, whose purpose is to define a better version of the SEARCH response defined in RFC 3501.

Section 3 defines the collected ABNF that replaces pieces of ABNF in the aforementioned RFCs. The collected ABNF got generalized to allow for easier future extensibility.

1.2. Conventions Used in This Document

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

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as defined in "Key words for use in RFCs to Indicate Requirement Levels" [KEYWORDS].

2. IMAP ABNF Extensions

This section is not normative. It provides some background on the intended use of different extensions and it gives some guidance about how future extensions should extend the described commands.

2.1. Optional Parameters with the SELECT/EXAMINE Commands

This document adds the ability to include one or more parameters with the IMAP SELECT (section 6.3.1 of [IMAP4]) or EXAMINE (section 6.3.2 of [IMAP4]) commands, to turn on or off certain standard behaviors, or to add new optional behaviors required for a particular extension.

There are two possible modes of operation:

- A global state change where a single use of the optional parameter will affect the session state from that time on, irrespective of subsequent SELECT/EXAMINE commands.

- A per-mailbox state change that will affect the session only for the duration of the new selected state. A subsequent SELECT/EXAMINE without the optional parameter will cancel its effect for the newly selected mailbox.

Optional parameters to the SELECT or EXAMINE commands are added as a parenthesized list of attribute/value pairs, and appear after the mailbox name in the standard SELECT or EXAMINE command. The order of individual parameters is arbitrary. A parameter value is optional

and may consist of atoms, strings, or lists in a specific order. If the parameter value is present, it always appears in parentheses (*). Any parameter not defined by extensions that the server supports must be rejected with a BAD response.

Example:

```
C: a SELECT INBOX (ANNOTATE)
S: ...
S: a OK SELECT complete
```

In the above example, a single parameter is used with the SELECT command.

Example:

```
C: a EXAMINE INBOX (ANNOTATE RESPONSES ("UID Responses")
   CONDSTORE)
S: ...
S: a OK EXAMINE complete
```

In the above example, three parameters are used with the EXAMINE command. The second parameter consists of two items: an atom "RESPONSES" followed by a quoted string.

Example:

```
C: a SELECT INBOX (BLURDYBLOOP)
S: a BAD Unknown parameter in SELECT command
```

In the above example, a parameter not supported by the server is used. This results in the BAD response from the server.

(*) - if a parameter has a mandatory value, which can always be represented as a number or a sequence-set, the parameter value does not need the enclosing (). See ABNF for more details.

2.2. Extended CREATE Command

```
Arguments:  mailbox name
            OPTIONAL list of CREATE parameters

Responses:  no specific responses for this command

Result:     OK - create completed
            NO - create failure: cannot create mailbox with
                 that name
            BAD - argument(s) invalid
```

This document adds the ability to include one or more parameters with the IMAP CREATE command (see section 6.3.3 of [IMAP4]), to turn on or off certain standard behaviors, or to add new optional behaviors required for a particular extension. No CREATE parameters are defined in this document.

Optional parameters to the CREATE command are added as a parenthesized list of attribute/value pairs after the mailbox name. The order of individual parameters is arbitrary. A parameter value is optional and may consist of atoms, strings, or lists in a specific order. If the parameter value is present, it always appears in parentheses (*). Any parameter not defined by extensions that the server supports must be rejected with a BAD response.

(*) - if a parameter has a mandatory value, which can always be represented as a number or a sequence-set, the parameter value does not need the enclosing (). See ABNF for more details.

2.3. Extended RENAME Command

```
Arguments:  existing mailbox name
            new mailbox name
            OPTIONAL list of RENAME parameters

Responses:  no specific responses for this command

Result:     OK - rename completed
            NO - rename failure: cannot rename mailbox with
                 that name, cannot rename to mailbox with
                 that name, etc.
            BAD - argument(s) invalid
```

This document adds the ability to include one or more parameters with the IMAP RENAME command (see section 6.3.5 of [IMAP4]), to turn on or off certain standard behaviors, or to add new optional behaviors required for a particular extension. No RENAME parameters are defined in this document.

Optional parameters to the RENAME command are added as a parenthesized list of attribute/value pairs after the new mailbox name. The order of individual parameters is arbitrary. A parameter value is optional and may consist of atoms, strings, or lists in a specific order. If the parameter value is present, it always appears in parentheses (*). Any parameter not defined by extensions that the server supports must be rejected with a BAD response.

(*) - if a parameter has a mandatory value, which can always be represented as a number or a sequence-set, the parameter value does not need the enclosing (). See ABNF for more details.

2.4. Extensions to FETCH and UID FETCH Commands

Arguments: sequence set
message data item names or macro
OPTIONAL fetch modifiers

Responses: untagged responses: FETCH

Result: OK - fetch completed
NO - fetch error: cannot fetch that data
BAD - command unknown or arguments invalid

This document extends the syntax of the FETCH and UID FETCH commands (see section 6.4.5 of [IMAP4]) to include optional FETCH modifiers. No fetch modifiers are defined in this document.

The order of individual modifiers is arbitrary. Each modifier is an attribute/value pair. A modifier value is optional and may consist of atoms and/or strings and/or lists in a specific order. If the modifier value is present, it always appears in parentheses (*). Any modifiers not defined by extensions that the server supports must be rejected with a BAD response.

(*) - if a modifier has a mandatory value, which can always be represented as a number or a sequence-set, the modifier value does not need the enclosing (). See ABNF for more details.

2.5. Extensions to STORE and UID STORE Commands

Arguments: message set
OPTIONAL store modifiers
message data item name
value for message data item

Responses: untagged responses: FETCH

Result: OK - store completed
NO - store error: cannot store that data
BAD - command unknown or arguments invalid

This document extends the syntax of the STORE and UID STORE commands (see section 6.4.6 of [IMAP4]) to include optional STORE modifiers. No store modifiers are defined in this document.

The order of individual modifiers is arbitrary. Each modifier is an attribute/value pair. A modifier value is optional and may consist of atoms and/or strings and/or lists in a specific order. If the modifier value is present, it always appears in parentheses (*). Any modifiers not defined by extensions that the server supports must be rejected with a BAD response.

(*) - if a modifier has a mandatory value, which can always be represented as a number or a sequence-set, the modifier value does not need the enclosing (). See ABNF for more details.

2.6. Extensions to SEARCH Command

2.6.1. Extended SEARCH Command

Arguments: OPTIONAL result specifier
OPTIONAL [CHARSET] specification
searching criteria (one or more)

Responses: REQUIRED untagged response: SEARCH (*)

Result: OK - search completed
NO - search error: cannot search that [CHARSET] or criteria
BAD - command unknown or arguments invalid

This section updates definition of the SEARCH command described in section 6.4.4 of [IMAP4].

The SEARCH command is extended to allow for result options. This document does not define any result options.

The order of individual options is arbitrary. Individual options may contain parameters enclosed in parentheses (**). If an option has parameters, they consist of atoms and/or strings and/or lists in a specific order. Any options not defined by extensions that the server supports must be rejected with a BAD response.

(*) - An extension to the SEARCH command may require another untagged response, or no untagged response to be returned. Section 2.6.2 defines a new ESEARCH untagged response that replaces the SEARCH untagged response. Note that for a given extended SEARCH command the SEARCH and ESEARCH responses SHOULD be mutually exclusive, i.e., only one of them should be returned.

(**) - if an option has a mandatory parameter, which can always be represented as a number or a sequence-set, the option parameter does not need the enclosing (). See ABNF for more details.

2.6.2. ESEARCH untagged response

Contents: one or more search-return-data pairs

The ESEARCH response SHOULD be sent as a result of an extended SEARCH or UID SEARCH command specified in section 2.6.1.

The ESEARCH response starts with an optional search correlator. If it is missing, then the response was not caused by a particular IMAP command, whereas if it is present, it contains the tag of the command that caused the response to be returned.

The search correlator is followed by an optional UID indicator. If this indicator is present, all data in the ESEARCH response refers to UIDs, otherwise all returned data refers to message numbers.

The rest of the ESEARCH response contains one or more search data pairs. Each pair starts with unique return item name, followed by a space and the corresponding data. Search data pairs may be returned in any order. Unless specified otherwise by an extension, any return item name SHOULD appear only once in an ESEARCH response.

Example: S: * ESEARCH UID COUNT 5 ALL 4:19,21,28

Example: S: * ESEARCH (TAG "a567") UID COUNT 5 ALL 4:19,21,28

Example: S: * ESEARCH COUNT 5 ALL 1:17,21

2.7. Extensions to APPEND Command

The IMAP BINARY extension [BINARY] extends the APPEND command to allow a client to append data containing NULs by using the <literal8> syntax. The ABNF was rewritten to allow for easier extensibility by IMAP extensions. This document hasn't specified any semantical changes to the [BINARY] extension.

In addition, the non-terminal "literal8" defined in [BINARY] got extended to allow for non-synchronizing literals if both [BINARY] and [LITERAL+] extensions are supported by the server.

The IMAP MULTIAPPEND extension [MULTIAPPEND] extends the APPEND command to allow a client to append multiple messages atomically. This document defines a common syntax for the APPEND command that takes into consideration syntactic extensions defined by both [BINARY] and [MULTIAPPEND] extensions.

3. 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 [IMAP4].

Except as noted otherwise, all alphabetic characters are case-insensitive. The use of uppercase or lowercase characters to define token strings is for editorial clarity only. Implementations MUST accept these strings in a case-insensitive fashion.

```
append          = "APPEND" SP mailbox 1*append-message
                  ;; only a single append-message may appear
                  ;; if MULTIAPPEND [MULTIAPPEND] capability
                  ;; is not present

append-message  = append-opts SP append-data

append-ext      = append-ext-name SP append-ext-value
                  ;; This non-terminal define extensions to
                  ;; to message metadata.

append-ext-name = tagged-ext-label

append-ext-value= tagged-ext-val
                  ;; This non-terminal shows recommended syntax
                  ;; for future extensions.


append-data     = literal / literal8 / append-data-ext

append-data-ext = tagged-ext
                  ;; This non-terminal shows recommended syntax
                  ;; for future extensions,
                  ;; i.e., a mandatory label followed
                  ;; by parameters.

append-opts     = [SP flag-list] [SP date-time] *(SP append-ext)
                  ;; message metadata

charset         = atom / quoted
                  ;; Exact syntax is defined in [CHARSET].

create          = "CREATE" SP mailbox
                  [create-params]
                  ;; Use of INBOX gives a NO error.
```

```
   create-params   = SP "(" create-param *( SP create-param) ")"

   create-param-name = tagged-ext-label

   create-param      = create-param-name [SP create-param-value]

   create-param-value= tagged-ext-val
                     ;; This non-terminal shows recommended syntax
                     ;; for future extensions.


   esearch-response  = "ESEARCH" [search-correlator] [SP "UID"]
                        *(SP search-return-data)
                      ;; Note that SEARCH and ESEARCH responses
                      ;; SHOULD be mutually exclusive,
                      ;; i.e., only one of the response types
                      ;; should be
                      ;; returned as a result of a command.


   examine         = "EXAMINE" SP mailbox [select-params]
                     ;; modifies the original IMAP EXAMINE command
                     ;; to accept optional parameters

   fetch           = "FETCH" SP sequence-set SP ("ALL" / "FULL" /
                     "FAST" / fetch-att /
                     "(" fetch-att *(SP fetch-att) ")")
                     [fetch-modifiers]
                     ;; modifies the original IMAP4 FETCH command to
                     ;; accept optional modifiers

   fetch-modifiers = SP "(" fetch-modifier *(SP fetch-modifier) ")"

   fetch-modifier  = fetch-modifier-name [ SP fetch-modif-params ]

   fetch-modif-params  = tagged-ext-val
                     ;; This non-terminal shows recommended syntax
                     ;; for future extensions.

   fetch-modifier-name = tagged-ext-label

   literal8        = "~{" number ["+"] "}" CRLF *OCTET
                      ;; A string that might contain NULs.
                      ;; <number> represents the number of OCTETs
                      ;; in the response string.
                      ;; The "+" is only allowed when both LITERAL+ and
                      ;; BINARY extensions are supported by the server.
```

```
   mailbox-data      =/ Namespace-Response /
                        esearch-response

   Namespace         = nil / "(" 1*Namespace-Descr ")"

   Namespace-Command = "NAMESPACE"

   Namespace-Descr   = "(" string SP
                         (DQUOTE QUOTED-CHAR DQUOTE / nil)
                          *(Namespace-Response-Extension) ")"

   Namespace-Response-Extension = SP string SP
                     "(" string *(SP string) ")"

   Namespace-Response = "NAMESPACE" SP Namespace
                        SP Namespace SP Namespace
         ;; This response is currently only allowed
         ;; if the IMAP server supports [NAMESPACE].
         ;; The first Namespace is the Personal Namespace(s)
         ;; The second Namespace is the Other Users' Namespace(s)
         ;; The third Namespace is the Shared Namespace(s)

   rename          = "RENAME" SP mailbox SP mailbox
                     [rename-params]
                     ;; Use of INBOX as a destination gives
                     ;; a NO error, unless rename-params
                     ;; is not empty.

   rename-params     = SP "(" rename-param *( SP rename-param) ")"

   rename-param      = rename-param-name [SP rename-param-value]

   rename-param-name = tagged-ext-label

   rename-param-value= tagged-ext-val
                     ;; This non-terminal shows recommended syntax
                     ;; for future extensions.


   response-data   = "*" SP response-payload CRLF

   response-payload= resp-cond-state / resp-cond-bye /
                     mailbox-data / message-data / capability-data

   search          = "SEARCH" [search-return-opts]
                     SP search-program

   search-correlator  = SP "(" "TAG" SP tag-string ")"
```

```
   search-program     = ["CHARSET" SP charset SP]
                        search-key *(SP search-key)
                        ;; CHARSET argument to SEARCH MUST be
                        ;; registered with IANA.

   search-return-data = search-modifier-name SP search-return-value
                        ;; Note that not every SEARCH return option
                        ;; is required to have the corresponding
                        ;; ESEARCH return data.

   search-return-opts = SP "RETURN" SP "(" [search-return-opt
                        *(SP search-return-opt)] ")"

   search-return-opt = search-modifier-name [SP search-mod-params]

   search-return-value = tagged-ext-val
                        ;; Data for the returned search option.
                        ;; A single "nz-number"/"number" value
                        ;; can be returned as an atom (i.e., without
                        ;; quoting).  A sequence-set can be returned
                        ;; as an atom as well.

   search-modifier-name = tagged-ext-label

   search-mod-params = tagged-ext-val
                     ;; This non-terminal shows recommended syntax
                     ;; for future extensions.


   select          = "SELECT" SP mailbox [select-params]
                     ;; modifies the original IMAP SELECT command to
                     ;; accept optional parameters

   select-params   = SP "(" select-param *(SP select-param) ")"

   select-param    = select-param-name [SP select-param-value]
                     ;; a parameter to SELECT may contain one or
                     ;; more atoms and/or strings and/or lists.

   select-param-name= tagged-ext-label

   select-param-value= tagged-ext-val
                     ;; This non-terminal shows recommended syntax
                     ;; for future extensions.


   status-att-list = status-att-val *(SP status-att-val)
                     ;; Redefines status-att-list from RFC 3501.
```

```
                    ;; status-att-val is defined in RFC 3501 errata

   status-att-val  = ("MESSAGES" SP number) /
                     ("RECENT" SP number) /
                     ("UIDNEXT" SP nz-number) /
                     ("UIDVALIDITY" SP nz-number) /
                     ("UNSEEN" SP number)
                     ;; Extensions to the STATUS responses
                     ;; should extend this production.
                     ;; Extensions should use the generic
                     ;; syntax defined by tagged-ext.

   store           = "STORE" SP sequence-set [store-modifiers]
                     SP store-att-flags
                     ;; extend [IMAP4] STORE command syntax
                     ;; to allow for optional store-modifiers

   store-modifiers =  SP "(" store-modifier *(SP store-modifier)
                       ")"

   store-modifier  = store-modifier-name [SP store-modif-params]

   store-modif-params = tagged-ext-val
                     ;; This non-terminal shows recommended syntax
                     ;; for future extensions.

   store-modifier-name = tagged-ext-label

   tag-string         = string
                       ;; tag of the command that caused
                       ;; the ESEARCH response, sent as
                       ;; a string.

   tagged-ext          = tagged-ext-label SP tagged-ext-val
                          ;; recommended overarching syntax for
                          ;; extensions

   tagged-ext-label    = tagged-label-fchar *tagged-label-char
                         ;; Is a valid RFC 3501 "atom".

   tagged-label-fchar  = ALPHA / "-" / "_" / "."

   tagged-label-char   = tagged-label-fchar / DIGIT / ":"
```

```
tagged-ext-comp     = astring /
                      tagged-ext-comp *(SP tagged-ext-comp) /
                      "(" tagged-ext-comp ")"
                       ;; Extensions that follow this general
                       ;; syntax should use nstring instead of
                       ;; astring when appropriate in the context
                       ;; of the extension.
                       ;; Note that a message set or a "number"
                       ;; can always be represented as an "atom".
                       ;; An URL should be represented as
                       ;; a "quoted" string.

tagged-ext-simple   = sequence-set / number

tagged-ext-val      = tagged-ext-simple /
                      "(" [tagged-ext-comp] ")"
```

4. Security Considerations

This document updates ABNF in RFCs 2088, 2342, 3501, 3502, and 3516. The updated documents must be consulted for security considerations for the extensions that they define.

As a protocol gets more complex, parser bugs become more common including buffer overflow, denial of service, and other common security coding errors. To the extent that this document makes the parser more complex, it makes this situation worse. To the extent that this document makes the parser more consistent and thus simpler, the situation is improved. The impact will depend on how many deployed IMAP extensions are consistent with this document. Implementers are encouraged to take care of these issues when extending existing implementations. Future IMAP extensions should strive for consistency and simplicity to the greatest extent possible.

Extensions to IMAP commands that are permitted in NOT AUTHENTICATED state are more sensitive to these security issues due to the larger possible attacker community prior to authentication, and the fact that some IMAP servers run with elevated privileges in that state. This document does not extend any commands permitted in NOT AUTHENTICATED state. Future IMAP extensions to commands permitted in NOT AUTHENTICATED state should favor simplicity over consistency or extensibility.

5. Normative References

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

[IMAP4] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

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

[CHARSET] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

[MULTIAPPEND] Crispin, M., "Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension", RFC 3502, March 2003.

[NAMESPACE] Gahrns, M. and C. Newman, "IMAP4 Namespace", RFC 2342, May 1998.

[LITERAL+] Myers, J., "IMAP4 non-synchronizing literals", RFC 2088, January 1997.

[BINARY] Nerenberg, L., "IMAP4 Binary Content Extension", RFC 3516, April 2003.

6. Acknowledgements

This documents is based on ideas proposed by Pete Resnick, Mark Crispin, Ken Murchison, Philip Guenther, Randall Gellens, and Lyndon Nerenberg.

However, all errors and omissions must be attributed to the authors of the document.

Thanks to Philip Guenther, Dave Cridland, Mark Crispin, Chris Newman, Elwyn Davies, and Barry Leiba for comments and corrections.

literal8 syntax was taken from RFC 3516.

Authors' Addresses

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

EMail: Alexey.Melnikov@isode.com

Cyrus Daboo

EMail: cyrus@daboo.name

Full Copyright Statement

Intellectual Property

Acknowledgement

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