

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 1, 2019

S. Brandt  
Verizon  
June 30, 2018

IMAP REPLACE Extension  
draft-brandt-imap-replace-03

Abstract

This document defines an IMAP extension which can be used to replace an existing message in a message store with a new message. Message replacement is a common operation for clients that automatically save drafts or notes as a user composes them.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on January 1, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Conventions Used in This Document . . . . .	2
2. Overview . . . . .	2
3. REPLACE and UID REPLACE . . . . .	3
3.1. Advertising Support for REPLACE . . . . .	3
3.2. REPLACE Command . . . . .	3
3.3. UID REPLACE Command . . . . .	4
3.4. Semantics of REPLACE and UID REPLACE . . . . .	4
3.5. IMAP State Diagram Impacts . . . . .	5
4. Interaction with other extensions . . . . .	6
4.1. RFC 4314, ACL . . . . .	6
4.2. RFC 4469, CATENATE . . . . .	6
4.3. RFC 4315, UIDPLUS . . . . .	8
4.4. RFC 6785, IMAP Events in Sieve . . . . .	8
4.5. RFC 7162, CONDSTORE/QRESYNC . . . . .	8
5. Formal Syntax . . . . .	8
6. Security Considerations . . . . .	9
7. IANA Considerations . . . . .	9
8. Acknowledgements . . . . .	9
9. References . . . . .	9
9.1. Normative References . . . . .	9
9.2. Informative References . . . . .	10
Author's Address . . . . .	10

## 1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Formal syntax is defined by [RFC5234].

Example lines prefaced by "C:" are sent by the client and ones prefaced by "S:" by the server.

## 2. Overview

This document defines an IMAP [RFC3501] extension to facilitate replacing an existing message with a new one. This is accomplished by defining a new REPLACE command and extending the UID command to allow UID REPLACE.

Since there is no replace function in the base IMAP specification, clients have instead had to use a combination of three separate commands issued in serial fashion; APPEND, STORE, EXPUNGE. Pipelining of these three commands is not recommended since failure



of any individual command should prevent subsequent commands from being executed lest the original message version be lost.

Because of the non-atomic nature of the existing sequence, interruptions can leave messages in intermediate states which can be seen and acted upon by other clients. Such interruptions can also strand older revisions of messages, thereby forcing the user to manually clean up multiple revisions of the same message in order to avoid wasteful quota consumption. Additionally, the existing sequence can fail on APPEND due to an over-quota condition even though the subsequent STORE/EXPUNGE would free up enough space for the newly revised message. And finally, server efficiencies may be possible with a single logical message replacement operation as compared to the existing APPEND/STORE/EXPUNGE sequence.

In its simplest form, the REPLACE command is a single-command encapsulation of APPEND, STORE +flags \DELETED and UID EXPUNGE for a message, except that it avoids any of the quota implications or intermediate states associated with the 3 command sequence. In handling a REPLACE command, a server MUST NOT generate a response code for the STORE +flags \DELETED portion of the sequence. Additionally, servers supporting the REPLACE command MUST NOT infer any inheritance of content, flags, or annotations from the message being replaced. Finally, the replaced and replacing messages SHOULD NOT be present in the mailbox at the same time.

### 3. REPLACE and UID REPLACE

#### 3.1. Advertising Support for REPLACE

Servers that implement the REPLACE extension will return "REPLACE" as one of the supported capabilities in the CAPABILITY command response.

#### 3.2. REPLACE Command

Arguments: message sequence number  
          mailbox name  
          OPTIONAL flag parenthesized list  
          OPTIONAL date/time string  
          message literal

Responses: no specific responses for this command

Result: OK - replace completed  
        NO - replace error; can't remove specified message  
            or can't add new message content  
        BAD - command unknown or arguments invalid



## Example:

```
C: A003 REPLACE 4 Drafts (\Seen \Draft) {312}
S: + Ready for literal data
C: Date: Thu, 1 Jan 2015 00:05:00 -0500 (EST)
C: From: Fritz Schmidt <fritz.ze@example.org>
C: Subject: happy new year !!
C: To: miss.mitzzy@example.org
C: Message-Id: <B238822388-0100000@example.org>
C: MIME-Version: 1.0
C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
C:
C: Just saw the best fireworks show. Wish you were here.
C:
S: * 5 EXISTS
S: * 4 EXPUNGE
S: A003 OK [APPENDUID 1 2000] Replace completed
```

## 3.3. UID REPLACE Command

This extends the first form of the UID command (see [RFC3501] Section 6.4.8) to add the REPLACE command defined above as a valid argument. This form of REPLACE uses a UID rather than sequence number as its first parameter.

## Example:

```
C: A004 UID REPLACE 2000 Drafts (\Seen \Draft) {350}
S: + Ready for literal data
C: Date: Thu, 1 Jan 2015 00:06:00 -0500 (EST)
C: From: Fritz Schmidt <fritz.ze@example.org>
C: Subject: happy new year !!
C: To: miss.mitzzy@example.org
C: Message-Id: <B238822389-0100000@example.org>
C: MIME-Version: 1.0
C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
C:
C: Just saw the best fireworks show. Wish you were here.
C: Hopefully next year you can join us.
C:
S: * 5 EXISTS
S: * 4 EXPUNGE
S: A004 OK [APPENDUID 1 2001] Replace completed
```

## 3.4. Semantics of REPLACE and UID REPLACE

The REPLACE and UID REPLACE commands take five arguments: a message identifier, a named mailbox, an optional parenthesized flag list, an optional message date/time string, and a message literal. The message literal will be appended to the named mailbox, and the



message specified by the message identifier will be removed from the selected mailbox. These operations will appear to the client as a single action. This has the same effect as the following sequence:

1. APPEND
2. [UID] STORE +FLAGS.SILENT \DELETED
3. UID EXPUNGE

In the cited sequence, the quota implications of the APPEND are evaluated within the context of the pending EXPUNGE so that only the net quota consumption is considered. Additionally, the EXPUNGE portion of the sequence only applies to the specified message, not all messages flagged as \Deleted.

Although the effect of REPLACE is identical to the steps above, the semantics are not identical; similar to MOVE [RFC6851], the intermediate states produced do not occur, and the response codes are different. In particular, the response codes for APPEND and EXPUNGE will be returned while those for the STORE operation MUST NOT be generated.

When an error occurs while processing REPLACE or UID REPLACE, the server MUST NOT leave the selected mailbox in an inconsistent state; any untagged EXPUNGE response MUST NOT be sent until all actions are successfully completed.

While it may be common for the named mailbox argument to match the selected mailbox for the common use case of replacing a draft, the REPLACE extension intentionally does not require the two to be the same. As an example, it's possible to use the REPLACE command to replace a message in the \Drafts special-use mailbox with a message in the \Sent special-use mailbox following message submission.

Because of the similarity of REPLACE to APPEND, extensions that affect APPEND affect REPLACE in the same way. Response codes such as TRYCREATE (see [RFC3501] Section 6.3.11), along with those defined by extensions, are sent as appropriate. See Section 4 for more information about how REPLACE interacts with other IMAP extensions.

### 3.5. IMAP State Diagram Impacts

Unlike the APPEND command which is valid in the authenticated state, the REPLACE and UID REPLACE commands MUST only be valid in the selected state. This difference from APPEND is necessary since REPLACE operates on message sequence numbers.



#### 4. Interaction with other extensions

This section describes how REPLACE interacts with some other IMAP extensions.

##### 4.1. RFC 4314, ACL

The ACL rights [RFC4314] required for UID REPLACE are the union of the ACL rights required for UID STORE and UID EXPUNGE in the current mailbox, and APPEND in the target mailbox.

##### 4.2. RFC 4469, CATENATE

Servers supporting both REPLACE and CATENATE [RFC4469] MUST support the additional append-data and resp-text-code elements defined the Formal Syntax section of RFC4469 in conjunction with the REPLACE command. When combined with CATENATE, REPLACE can become a quite efficient way for message manipulation.



## Example:

User composes message and attaches photo

```
-----
C: A010 APPEND Drafts (\Seen \Draft) {1201534}
S: + Ready for literal data
C: Date: Thu, 1 Jan 2015 00:10:00 -0500 (EST)
C: From: Fritz Schmidt <fritz.ze@example.org>
C: Message-ID: <B238822388-0100003@example.org>
C: MIME-Version: 1.0
C: Content-Type: multipart/mixed;
C:           boundary="-----030305060306060609050804"
C:
C: -----030305060306060609050804
C: Content-Type: text/plain; charset=utf-8; format=flowed
C: Content-Transfer-Encoding: 7bit
C:
C: Here is picture from the fireworks
C:
C: Yours...
C: Fritz
C:
C: -----030305060306060609050804
C: Content-Type: image/jpeg;
C:           name="Fireworks.jpg"
C: Content-Transfer-Encoding: base64
C: Content-Disposition: attachment;
C:           filename="Fireworks.jpg"
C:
C: <large base64 encoded part goes here>
C:
C: -----030305060306060609050804--
S: A010 OK [APPENDUID 1 3002] APPEND complete
```

User completes message with To: and Subject: fields

```
-----
C: A011 UID REPLACE 3002 Drafts CATENATE (TEXT {71}
S: + Ready for literal data
C: To: Mitzy <miss.mitzy@example.org>
C: Subject: My view of the fireworks
C: URL "/Drafts/;UID=3002")
S: * 5 EXISTS
S: * 4 EXPUNGE
S: A011 OK [APPENDUID 1 3003] REPLACE completed
```



### 4.3. RFC 4315, UIDPLUS

Servers supporting both REPLACE and UIDPLUS [RFC4315] SHOULD send APPENDUID in response to a UID REPLACE command. For additional information see section 3 of RFC4315. Servers implementing REPLACE and UIDPLUS are also advised to send the APPENDUID response code in an untagged OK before sending the EXPUNGE or replaced responses. (Sending the APPENDUID in the tagged OK, as described in the UIDPLUS specification means that the client first receives an EXPUNGE for a message and afterwards APPENDUID for the new message. It can be unnecessarily difficult to process that sequence usefully.)

#### 4.4. RFC 6785, IMAP Events in Sieve

REPLACE applies to IMAP events in Sieve [RFC6785] in the same way that APPEND does. Therefore, REPLACE can cause a Sieve script to be invoked with the `imap.cause` set to "APPEND". Because the intermediate state of `STORE +FLAGS.SILENT \DELETED` is not exposed by REPLACE, no action will be taken that results in a `imap.cause` of `FLAG`.

#### 4.5. RFC 7162, CONDSTORE/QRESYNC

Servers implementing both REPLACE and CONDSTORE/QRESYNC [RFC7162] MUST treat the message being replaced as if it were being removed with a UID EXPUNGE command. Sections 3.2.9 and 3.2.10 of RFC 7162 are particularly relevant for this condition.

## 5. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234]. [RFC3501] defines the non-terminals "capability", "command-select", "mailbox", and "sequence-number". [RFC4466] defines the non-terminal "append-message".

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

```
capability    =/ "REPLACE"
```

```
command-select =/ replace
```

```
replace          = "REPLACE" SP seq-number SP mailbox append-message
```

```
uid          = "UID" SP (copy / fetch/ search / store / move /
                        replace)
```



## 6. Security Considerations

This document is believed to add no security problems beyond those that may already exist with the base IMAP specification.

## 7. IANA Considerations

The IANA is requested to add REPLACE to the "IMAP 4 Capabilities" registry, <http://www.iana.org/assignments/imap4-capabilities>.

## 8. Acknowledgements

The author would like to thank the participants of IMAPEXT with particular thanks to Arnt Gulbrandsen, Alexey Melkinov, Chris Newman, and Bron Gondwana for their specific contributions.

## 9. References

### 9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <<https://www.rfc-editor.org/info/rfc3501>>.
- [RFC4314] Melnikov, A., "IMAP4 Access Control List (ACL) Extension", RFC 4314, DOI 10.17487/RFC4314, December 2005, <<https://www.rfc-editor.org/info/rfc4314>>.
- [RFC4315] Crispin, M., "Internet Message Access Protocol (IMAP) - UIDPLUS extension", RFC 4315, DOI 10.17487/RFC4315, December 2005, <<https://www.rfc-editor.org/info/rfc4315>>.
- [RFC4466] Melnikov, A. and C. Daboo, "Collected Extensions to IMAP4 ABNF", RFC 4466, DOI 10.17487/RFC4466, April 2006, <<https://www.rfc-editor.org/info/rfc4466>>.
- [RFC4469] Resnick, P., "Internet Message Access Protocol (IMAP) CATENATE Extension", RFC 4469, DOI 10.17487/RFC4469, April 2006, <<https://www.rfc-editor.org/info/rfc4469>>.



- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6785] Leiba, B., "Support for Internet Message Access Protocol (IMAP) Events in Sieve", RFC 6785, DOI 10.17487/RFC6785, November 2012, <<https://www.rfc-editor.org/info/rfc6785>>.
- [RFC7162] Melnikov, A. and D. Cridland, "IMAP Extensions: Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC)", RFC 7162, DOI 10.17487/RFC7162, May 2014, <<https://www.rfc-editor.org/info/rfc7162>>.

## 9.2. Informative References

- [RFC6851] Gulbrandsen, A. and N. Freed, Ed., "Internet Message Access Protocol (IMAP) - MOVE Extension", RFC 6851, DOI 10.17487/RFC6851, January 2013, <<https://www.rfc-editor.org/info/rfc6851>>.

## Author's Address

Stuart Brandt  
Verizon  
22001 Loudoun County Parkway  
Ashburn, VA 20147  
USA

Email: [stujenerin@aol.com](mailto:stujenerin@aol.com)



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: October 18, 2018

M. Slusarz  
Open-Xchange Inc.  
April 16, 2018

IMAP4 Extension: Message Snippet Generation  
draft-slusarz-imap-fetch-snippet-00

Abstract

This document specifies an IMAP protocol extension which allows a client to request that a server provide an abbreviated representation of a message (a snippet of text) that can be used by a client to provide a useful contextual preview of the message contents.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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."

This Internet-Draft will expire on October 18, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Introduction . . . . .	2
2. Conventions Used In This Document . . . . .	3
3. FETCH Data Item . . . . .	3
3.1. Command . . . . .	3
3.2. Response . . . . .	4
4. SNIPPET Algorithms . . . . .	4
4.1. FUZZY . . . . .	4
5. SNIPPET Priority Modifiers . . . . .	5
5.1. LAZY . . . . .	5
6. Examples . . . . .	6
7. Formal Syntax . . . . .	8
8. Acknowledgements . . . . .	9
9. IANA Considerations . . . . .	9
10. Security Considerations . . . . .	9
11. References . . . . .	9
11.1. Normative References . . . . .	9
11.2. Informative References . . . . .	10
Appendix A. Change History (To be removed by RFC Editor before publication) . . . . .	10
Author's Address . . . . .	10

## 1. Introduction

Many modern mail clients display small extracts of the body text as an aid to allow a user to quickly decide whether they are interested in viewing the full message contents. Mail clients implementing the Internet Message Access Protocol (IMAP; RFC 3501 [RFC3501]) would benefit from a standardized, consistent way to generate these brief previews of messages (a "snippet").

Generation of snippets on the server has several benefits. First, it allows consistent representation of snippets across all clients. This standardized display can reduce user confusion when using multiple clients, as abbreviated message representations in clients will show identical message details.

Second, server-side snippet generation is more efficient. A client-based algorithm needs to issue, at a minimum, a FETCH BODYSTRUCTURE command in order to determine which MIME [RFC2045] body part(s) should be represented in the snippet. Subsequently, at least one FETCH BODY command may be needed to retrieve body data used in snippet generation. These FETCH commands cannot be pipelined since the BODYSTRUCTURE query must be parsed on the client before the list of parts to be retrieved via the BODY command(s) can be determined.



Additionally, it may be difficult to predict the amount of body data that must be retrieved to adequately represent the part via a snippet, therefore requiring inefficient fetching of excessive data in order to account for this uncertainty. For example, a snippet algorithm to display data contained in a text/html [RFC2854] part will likely strip the markup tags to obtain textual content. However, without fetching the entire content of the part, there is no way to guarantee that sufficient non-tag content will exist unless either 1) the entire part is retrieved or 2) an additional partial FETCH is executed when the client determines that it does not possess sufficient data from a previous partial FETCH to display an adequate representation of the snippet.

Finally, server generation allows caching in a centralized location. Using server generated snippets allows snippets to be generated globally once per message, and then cached indefinitely. Retrieval of message data may be expensive within a server, for example, so a server can be configured to reduce its storage retrieval load by pre-generating snippet data.

A server that supports the SNIPPET extension indicates this with one or more capability names consisting of "SNIPPET=" followed by a supported snippet algorithm name. This format provides for future upwards-compatible extensions and/or the ability to use locally-defined snippet algorithms.

## 2. Conventions Used In This Document

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 RFC 2119 [RFC2119].

"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. If a single "C:" or "S:" label applies to multiple lines, then the line breaks between those lines are for editorial clarity only and are not part of the actual protocol exchange.

## 3. FETCH Data Item

### 3.1. Command

To retrieve a snippet for a message, the "SNIPPET" FETCH attribute is used when issuing a FETCH command.



If no algorithm identifier is provided, the server decides which of its built-in algorithms to use to generate the snippet text.

Alternately, the client may explicitly indicate which algorithm(s) should be used in a parenthesized list after the SNIPPET attribute containing the name of the algorithm. These algorithms MUST be one of the algorithms identified as supported in the SNIPPET capability responses. If a client requests an algorithm that is unsupported, the server MUST return a tagged BAD response.

The order of the algorithms in the parenthesized list (from left to right) defines the client's priority decision. Duplicate algorithms in the list SHOULD be ignored. For purposes of duplicate detection, priority modifiers (Section 5) should be ignored. A server MUST honor a client's algorithm priority decision.

### 3.2. Response

The algorithm used by the server to generate the snippet is returned preceding the snippet string.

The server returns a variable-length string that is the generated snippet for that message.

A server SHOULD strive to generate the same string for a given message for each request. However, since snippets are understood to be a representation of the message data and not a canonical view of its contents, a client MUST NOT assume that a message snippet is immutable for a given message. This relaxed requirement permits a server to offer snippets as an option without requiring potentially burdensome storage and/or processing requirements to guarantee immutability for a use case that does not require this strictness.

If the snippet is not available, the server MUST return NIL as the SNIPPET response. A NIL response indicates to the client that snippet information MAY become available in a future SNIPPET FETCH request.

## 4. SNIPPET Algorithms

### 4.1. FUZZY

The FUZZY algorithm directs the server to use any internal algorithm it desires, subject to the below limitations, to generate a textual snippet for a message.

The FUZZY algorithm MUST be implemented by any server that supports the SNIPPET extension.



The generated string **MUST NOT** be content transfer encoded and **MUST** be encoded in UTF-8 [RFC3629].

The snippet text **MUST** be treated as text/plain MIME data by the client.

The server **SHOULD** limit the length of the snippet text to 100 characters. The server **MUST NOT** output snippet text longer than 200 characters.

The server **SHOULD** remove any formatting markup that exists in the original text.

If the FUZZY algorithm generates a snippet that is not based on the body content of the message and the LANGUAGE [RFC5255] extension is supported by the server, the snippet text **SHOULD** be generated according to the the language rules that apply to human-readable text.

## 5. SNIPPET Priority Modifiers

### 5.1. LAZY

The LAZY modifier directs the server to return the snippet representation only if that data can be returned without undue delay to the client.

This modifier allows a client to inform the server that snippet data is nice-to-have, but the server **SHOULD NOT** block the return of other FETCH information at the expense of generating the snippet data.

For example, a client displaying the initial mailbox listing to a user may want to display snippet information associated with messages in that listing. However, this information is secondary to providing the mailbox listing, with message details, and the client is willing to load any unavailable snippets in the background and display them as they are provided by the server. In this case, the client would use the LAZY modifier to the desired algorithm(s) to direct the server to only return pre-generated snippet data so that retrieval of the other FETCH information is not blocked by possibly expensive snippet generation.

The LAZY modifier **MUST** be implemented by any server that supports the SNIPPET extension.



## 6. Examples

Example 1: Requesting FETCH without explicit algorithm selection

```
C: A1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SNIPPET=FUZZY
S: A1 OK Capability command completed.
[...a mailbox is SELECTed...]
C: A2 FETCH 1 (RFC822.SIZE SNIPPET)
S: * 1 FETCH (RFC822.SIZE 20000 SNIPPET (FUZZY {61}
S: This is the first line of text from the first text part.
S: ))
S: A2 OK FETCH complete.
```

Example 2: Requesting FETCH with explicit algorithm selection

```
C: B1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SNIPPET=FUZZY
S: B1 OK Capability command completed.
[...a mailbox is SELECTed...]
C: B2 FETCH 1 (RFC822.SIZE SNIPPET (FUZZY))
S: * 1 FETCH (RFC822.SIZE 20000 SNIPPET (FUZZY {61}
S: This is the first line of text from the first text part.
S: ))
S: B2 OK FETCH complete.
```

Example 3: Requesting FETCH with invalid explicit algorithm selection

```
C: C1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SNIPPET=FUZZY
S: C1 OK Capability command completed.
[...a mailbox is SELECTed...]
C: C2 FETCH 1 (RFC822.SIZE SNIPPET (X-SNIPPET-ALGO))
S: C2 BAD FETCH contains invalid snippet algorithm name.
```



Example 4: Use explicit algorithm priority selection, with LAZY modifier, to obtain snippets during initial mailbox listing if readily available; otherwise, load snippets in background

```
C: D1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SNIPPET=FUZZY
S: D1 OK Capability command completed.
[...a mailbox is SELECTed...]
C: D2 FETCH 1:3 (ENVELOPE SNIPPET (LAZY=FUZZY))
S: * 1 FETCH (ENVELOPE ("Wed, 25 Oct 2017 15:03:11 +0000" [...])
  SNIPPET (FUZZY {61})
S: This is the first line of text from the first text part.
S: ))
S: * 2 FETCH (SNIPPET (FUZZY "") ENVELOPE
  ("Thu, 26 Oct 2017 12:17:23 +0000" [...]))
S: * 3 FETCH (ENVELOPE ("Fri, 27 Oct 2017 22:19:21 +0000" [...])
  SNIPPET (FUZZY NIL))
S: D2 OK FETCH completed.
[...Client knows that message 2 has a snippet that is empty;
  therefore, client only needs to request message 3 snippet again
  (e.g. in background)...]
C: D3 FETCH 3 (SNIPPET (FUZZY))
S: * 3 FETCH (SNIPPET (FUZZY {25})
S: First sentence of mail 3.
S: ))
S: D3 OK Fetch completed.
```



Example 5: Retrieve snippet information for search results within a single mailbox. Use SEARCHRES [RFC5182] extension to save a round-trip.

```
C: E1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SNIPPET=FUZZY SEARCHRES
S: E1 OK Capability command completed.
[...a mailbox is SELECTed...]
C: E2 SEARCH RETURN (SAVE) FROM "FOO"
C: E3 FETCH $ (UID SNIPPET (LAZY=FUZZY))
S: E2 OK SEARCH completed.
S: * 5 FETCH (UID 13 SNIPPET (FUZZY {9}
S: Snippet!
S: ))
S: * 9 FETCH (UID 23 SNIPPET (FUZZY NIL))
S: E3 OK FETCH completed.
[...Retrieve message 9 snippet in background...]
C: E4 UID FETCH 23 (SNIPPET (FUZZY))
S: * 9 FETCH (SNIPPET (FUZZY {17}
S: Another snippet!
S: ))
S: E4 OK FETCH completed.
```

## 7. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in ABNF [RFC5234]. It includes definitions from IMAP [RFC3501].

```
capability          =/ "SNIPPET=FUZZY"

fetch-att           =/ "SNIPPET" [SP "(" snippet-alg-fetch *(SP
                           snippet-alg-fetch) ")"]

msg-att-dynamic     =/ "SNIPPET" SP "(" snippet-alg SP nstring ")"

snippet-alg         = "FUZZY" / snippet-alg-ext

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

snippet-alg-fetch   = snippet-alg / snippet-mod "=" snippet-alg

snippet-mod         = "LAZY" / snippet-mod-ext

snippet-mod-ext     = atom ; New priority modifiers MUST be
                           ; registered with IANA
```



## 8. Acknowledgements

The author would like to thank the following people for their comments and contributions to this document: Stephan Bosch, Teemu Huovila, Jeff Sipek, Timo Sirainen, Steffen Templin, and Aki Tuomi.

## 9. IANA Considerations

IMAP4 [RFC3501] capabilities are registered by publishing a standards track or IESG-approved experimental RFC. The registry is currently located at:

<http://www.iana.org/assignments/imap-capabilities>

This document requests that IANA adds the "SNIPPET=FUZZY" capability to the IMAP4 [RFC3501] capabilities registry.

## 10. Security Considerations

There are no known additional security issues with this extension beyond those described in the base protocol described in IMAP4 [RFC3501].

## 11. References

### 11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <<https://www.rfc-editor.org/info/rfc3501>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC5255] Newman, C., Gulbrandsen, A., and A. Melnikov, "Internet Message Access Protocol Internationalization", RFC 5255, DOI 10.17487/RFC5255, June 2008, <<https://www.rfc-editor.org/info/rfc5255>>.



## 11.2. Informative References

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <<https://www.rfc-editor.org/info/rfc2045>>.
- [RFC2854] Connolly, D. and L. Masinter, "The 'text/html' Media Type", RFC 2854, DOI 10.17487/RFC2854, June 2000, <<https://www.rfc-editor.org/info/rfc2854>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC5182] Melnikov, A., "IMAP Extension for Referencing the Last SEARCH Result", RFC 5182, DOI 10.17487/RFC5182, March 2008, <<https://www.rfc-editor.org/info/rfc5182>>.

## Appendix A. Change History (To be removed by RFC Editor before publication)

Changes from draft-slusarz-imap-fetch-snippet-00:

- o TODO

## Author's Address

Michael Slusarz  
Open-Xchange Inc.  
Denver, Colorado  
US

Email: [michael.slusarz@open-xchange.com](mailto:michael.slusarz@open-xchange.com)



INTERNET-DRAFT  
<draft-yu-imap-client-id-07.txt>  
Intended Status: Standards Track  
Expires May 18, 2022

Y. Deion  
LinuxMagic

November 18, 2021

IMAP Service Extension for Client Identity  
<draft-yu-imap-client-id-07.txt>

Abstract

This document defines an Internet Message Access Protocol (IMAP) service extension called "CLIENTID" which provides a method for clients to indicate an identity to the server.

This identity is an additional token that may be used for security and/or informational purposes, and with it a server may optionally apply heuristics using this token.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. 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/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Introduction .....	2
2. Conventions Used in This Document .....	3
3. CLIENTID .....	3
3.1. CLIENTID Command .....	3
3.2. CLIENTID Arguments .....	3
3.3. Advertising the CLIENTID capability .....	4
3.4. Restrictions on the CLIENTID command .....	4
4. Formal Syntax .....	4
5. Discussion .....	5
5.1. Background .....	5
5.2. Applying heuristics to CLIENTID .....	6
5.3. Utility of CLIENTID .....	6
5.4. Use Cases of CLIENTID .....	7
5.5. Other IMAP Client Identifiers .....	8
5.6. Future Considerations .....	8
6. Client Identity Types .....	8
7. Examples .....	10
7.1. UUID as Client Identity .....	10
7.2. Malformed CLIENTID Command .....	11
7.3. Client Identity Without a TLS/SSL Session .....	11
7.4. Client Identity Leading to Rejection .....	11
8. Security Considerations .....	12
9. IANA Considerations .....	12
10. References .....	12
10.1. Normative References .....	12
Appendix A. CLIENTID Product Support .....	13
Contributors .....	13
Authors' Addresses .....	13

## 1. Introduction

The [IMAP] protocol and its extensions describe methods whereby an client may provide identity and/or authentication information to an IMAP server. However, these existing methods are subject to limitations and none offer a way to identify the IMAP client with absolute confidence. This document defines an IMAP service extension to provide an additional identity token which can represent the IMAP client with a higher degree of certainty when accessing the IMAP server.

Typically IMAP clients enter the authenticated state by using either the AUTHENTICATE or LOGIN command. IMAP servers are often subject to malicious clients attempting to use authorization credentials and/or identities not intended for their use (e.g. stolen credentials or brute force attacks). When such an attack is attempted, the IMAP server may be unable to identify the impersonation and restrict such an unintended use by someone other than the authorized user or said credentials. While there are ways to identify the source of the IMAP client such as its IP address, it would be useful if there was an additional way to uniquely identify the client in a method solely available across an encrypted channel.



Using the CLIENTID extension, an IMAP client can provide an additional identity token to the server called its "client identity". The client identity can provide unique characteristics about the client accessing the IMAP service and may be combined with existing identification mechanisms in order to identify the client. An IMAP server may then apply additional security policies using this identity such as restricting use of the service to clients presenting recognized client identities or only allowing use of authorized identities that match previously established client identities.

The CLIENTID extension is present in any IMAP implementation that returns "CLIENTID" as one of the supported capabilities to the CAPABILITY command.

## 2. Conventions Used in This Document

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].

Formal syntax is specified using [ABNF].

Example lines prefaced by "C:" are sent by the client and ones prefaced by "S:" by the server.

"Connection" refers to the entire sequence of client/server interaction from the initial establishment of the network connection until its termination.

## 3. CLIENTID

### 3.1. CLIENTID Command

Arguments: client identity type  
            client identity token

Responses: no specific responses for this command

Result: OK - clientid completed, client identity stored  
        BAD - command unknown or arguments invalid

Note that a valid CLIENTID command will never return the NO result because heuristics MUST NOT be applied to the CLIENTID arguments at this stage. Instead the client identity information SHOULD be stored and passed along to any and all [SASL] authentication mechanisms.

### 3.2. CLIENTID Arguments

The CLIENTID command takes the following two arguments:

1. client identity type: A string identifying the identity type the client is providing. It MUST be between 1 and 16 alphanumeric and dash characters.



2. client identity token: A string identifying the client. It MUST be between 1 and 128 printable characters.

The IMAP server MUST reject any CLIENTID command with badly formatted arguments. The IMAP server MUST accept the arguments from a valid CLIENTID command and SHOULD store it at the minimum for the remaining duration of the IMAP connection.

### 3.3. Advertising the CLIENTID capability

The CLIENTID capability is used to tell the IMAP client that the IMAP server supports the CLIENTID extension. However, certain conditions MUST be met before the IMAP server advertises the CLIENTID capability.

1. The IMAP server and IMAP client MUST negotiate encryption via STARTTLS/SSL or some other secure mechanism.
2. The IMAP server MUST be in the non-authenticated state.
3. The IMAP server MUST have the CLIENTID extension support enabled.

While all the conditions are met, the IMAP server MUST advertise the CLIENTID capability in all proceeding CAPABILITY commands.

### 3.4. Restrictions on the CLIENTID command

Under certain circumstances, the use of the CLIENTID command will be restricted:

1. Before the CLIENTID capability has been advertised, the IMAP server MUST reject any issued CLIENTID command and the IMAP client MUST NOT issue the CLIENTID command.
2. Outside of the non-authenticated state, the IMAP server MUST reject any CLIENTID command issued by the IMAP client and the IMAP client MUST NOT issue the CLIENTID command.
3. Once a valid CLIENTID command has been issued, the IMAP server MUST reject any further CLIENTID command issued by the IMAP client and the IMAP client MUST NOT issue any subsequent CLIENTID commands.

## 4. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form notation as specified in [ABNF]. [IMAP] defines the non-terminals "capability" and "command-nonauth".

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



```
capability      =/ "CLIENTID"

command-nonauth =/ client-id

client-id       = "CLIENTID" SP client-id-type SP client-id-token

client-id-type  = 1*16 ALPHA / DIGIT / "-"
                  ;; alphanumeric with dash character

client-id-token = 1*128 VCHAR
                  ;; any printable US-ASCII character
```

## 5. Discussion

### 5.1 Background

The historical standard of using the user and password combination as a means of authentication is no longer effective in this day and age with recent developments in the world.

1. ISPs transistioning to Carrier-grade NAT due to IPv4 address exhaustion placing multiple devices behind the same IP address.
2. Numerous large scale data breaches exposing millions of user and password combinations.
3. Continued propensity for user to use same simple passwords across multiple accounts.
4. Botnets growing larger and more sophisticated due to the profileration of IoT devices.

As a result, brute force attacks against web services have become increasingly effective as malicious actors have easy access to millions of email addresses, commonly used passwords and massive botnets while the safety practices of users have not improved.

The traditional methods of defending against these types of attacks like rate limiting and blocking by IP addresses are no longer viable without collateral damange as thousands of devices could potentially be behind the same IP address as more ISPs adopt the CGN/LSN/NAT444 standard, i.e. blocking an IP address due to the actions of a single malicious actor bears the risk of blocking legitimate users.

By introducing CLIENTID as another non-public factor to be used in tandem with the user and password combination, authentication becomes much more resilient against brute force attacks. The email addresses and passwords exposed from the data breaches will no longer be sufficient to authenticate. Rate limiting and blocking can be performed based on the CLIENTID such that only a subset of devices behind the same IP address gets blocked. CLIENTID would also be backwards compatible with existing authentication protocols encouraging adoption.



## 5.2. Applying heuristics to CLIENTID

This section discusses the possible heuristics that can be applied to the information that is presented via the CLIENTID command. This information includes whether a valid CLIENTID command was issued, the client identity type and the client identity token.

1. The IMAP server MAY choose to require that a successful CLIENTID command be issued or that a particular client identity type be presented before processing or accepting an authentication request.
2. The IMAP server MAY reject any authentication request not preceded with a client identity type that matches ACL's or rules as defined in the IMAP server.
3. An IMAP server MAY reject any authentication request preceded by a CLIENTID command that contains a client identity type or client identity token that the server chooses not to accept for any reason such as by policy.
4. An IMAP server MAY reject any authentication request preceded by a CLIENTID command that contains a client identity type or client identity token that the server has chosen to disable or revoke use of either temporarily or permanently.

The IMAP server SHOULD only ever reject an IMAP client based on CLIENTID information during or after the authentication process/handler. In the interest of limiting the amount of information being revealed, the rejection message SHOULD be as generic as possible and SHOULD NOT reveal any information on the heuristics.

Even if the client identity type and/or client identity token are not recognized, supported or permitted by the server and/or the owner of the authentication credentials, the presented information may still be useful for analysis.

## 5.3. Utility of CLIENTID

Regardless of how frowned upon, users commonly reuse authorization information (like the username and password pair) across multiple services. When one service is compromised, malicious actors can also gain access to other services where the user also used the same credentials. Based on this representative problem alone, the utility of CLIENTID as an additional layer of determining the rights to present such authorization information becomes quickly apparent.

The utility of CLIENTID may be seen by considering the following:

1. An IMAP server could recognize a device not historically known to have presented the authentication credentials before.
2. An IMAP server could restrict authentication from actors not presenting a valid CLIENTID, or an account holder that the IMAP



server provides service for could restrict authentication to only those devices that present valid CLIENTID.

3. An IMAP server could restrict authentication to only devices which present a CLIENTID containing a client type identifier which the account holder or operator of the server deems to be permitted. (Eg. Only allow vendor A's devices)
4. An IMAP server could alert an account holder that an attempt to present their authorization credentials came from an unknown, unrecognized, or different device.

However, this extends beyond just the restriction of authentication. While it might be argued that this can be served as a special form of SASL, by implementing this in the IMAP service itself, the IMAP service can choose before allowing a connection to be passed to a SASL implementation, allowing it to perform other heuristics, such as brute force attacks, more effeciently.

While 'forgery' and/or the use of random client identifier is possible, such behavior is also more readily detectable when a device identifier is presented.

1. The IMAP server, when faced with hundreds of devices behind the same IP address, during an attack can restrict authentication attempts to only connections presenting a valid client identifier token.
2. The IMAP server, during an attack, can restrict authentication to only historically known devices.
3. The IMAP server can differentiate between many different devices behind the same IP, and apply maximum connections per device, rather than maximum connections per IP.
4. While a person may present authentication credentials from many different geographical locations, eg, home, office, and travel, a single device will not in general be able to be in two geographical locations at the same time. The IMAP server will have new information to apply to threat detection heuristics, ie to treat the use of the same client indentifier token from two locations, as a possible brute force or forgery situation.

#### 5.4. Use Cases of CLIENTID

With CLIENTID the IMAP server has additional information it may use in its interactions with the client. It may:

1. Restrict use of an authorization tokens to a set of client identity token identities, thereby offering an added level of security. For example the use of authorization credentials may only be accompanied by a specified set of CLIENTID tokens and/or types for a specific account holder, or set of account holders



2. Identify that the same CLIENTID token is used to access multiple authorized identities, and restrict access to the IMAP service. For example a malicious client that has attempted to gain access using multiple authorization tokens may be identified through its unusual behavior.
3. Retain knowledge of CLIENTID tokens previously presented with specific authorization credentials, and if the token has not been previously seen, restrict access to the IMAP service.
4. Require that the IMAP client present a token such as a license key established outside of the IMAP session in order to make use of any authorized identity.
5. Apply different security policies to clients that provide a CLIENTID token versus those which do not. For example, provide clients providing such an identity with additional trust.
6. Ability to rate limit or block based on the presented client-identifier-token, when multiple devices use a shared IP address, without affecting other devices.
7. Ability to detect distributed and localized dictionary attacks and brute force attacks.
8. Use the client-identifier-token as a third factor to be passed to authentication methods. [SASL]

#### 5.5. Other IMAP Client Identifiers

The [IMAP] protocol and its extensions describe methods whereby an IMAP client may provide identity information to an IMAP server. Some of these identifiers are listed for contrast:

1. The client connection provides a source IP address associated with the IMAP session. This may be accompanied by a PTR record and/or GeoIP information.
2. The AUTHENTICATE and LOGIN command allows the client to present a user and/or password/authentication mechanism for an IMAP session.

#### 5.6. Future Considerations

In the future there may be a demand for being able to provide multiple CLIENTID commands with different client identity types. For instance, it may be desirable for a device to identify itself, both with a hardware device identifier, and a software identifier. We believe this to be out of scope, and can be accommodated with a special client-identifier-token which encapsulates both.

#### 6. Client Identity Types

This document does not specify any CLIENTID identity type that MUST



be supported. The client identity type is meant to be defined by the client implementation that is designed to access the IMAP server and protocol. For instance, many IMAP client software implementations already create a distinct UUID for each account. Some commercial email clients have a license key. Some physical devices that need to interact with IMAP might have a unique hardware ID. While there is no pre-defined list of client identity type defined by this RFC, and all IMAP servers should be prepared to accept any form of client identity type that conforms to the definition, it is suggested that IMAP client developers carefully consider the name of the client identity type. For example, rather than using a client identity type of UUID, consider the advantages of making it more distinct, e.g. "<product\_short\_code>UUID". This way the IMAP server can better record histories, eg the difference between say a Thunderbird generated unique id, and a Mutt generated unique id.

Some examples of identity type might be UUID, LICENSE, DEVICE\_ID, and/or COOKIE. It is expected that the most common types might be related to distinct UUID, LICENSEKEY, or HARDWAREID.

An IMAP server SHOULD NOT reject an unidentified CLIENTID type, except for specific policy use cases.

It is envisioned that in the future it will be useful to propose a set of standardized client-identity-type to help with validation, or to allow the IMAP server to apply ACL rules on expected types, this would be an extension to this RFC.

#### 1. UUID

UUID is a common practice to represent either a individual user, hardware device or software installation associated with a specific individual. The support of UUID enables existing UUID implementations to be used to semi-uniquely identify a device associated with an individual. A definition of the format should be considered. Otherwise non-standard UUID might be a separate type specific to the software implementation, for instance TBIRD-UUID.

#### 2. LICENSE

An IMAP client may find it useful to identify the license key of software it is using. Such licenses are typically crafted such that they are unique and useful to identify a software installation. This is more normally suited for a software designed for a single-user. While LICENSE could be standard type again, it might more more helpful to specify a vendor specific type such as BBLICENSEKEY.

#### 3. DEVICE\_ID

Many hardware devices are designed to be used by a single individual and already have an associated hardware device id.



While a standard type might be defined, it also might be more helpful to use a vendor specific type, such as ATOM-DEVICEID.

#### 4. COOKIE

While not guaranteed to be consistent many web applications are designed to access IMAP directly and may need to have a semi-unique identifier available as part of the web based transaction. It is assumed that COOKIE encompasses the group of web based tokens known to persist from session to session. A specific web based application can provide sufficient information in the actual client-identifier-token to differentiate between applications and or websites, and are convenient as they can be related to very specific domains, and are universally available to web application designers.

As a reminder, an IMAP server SHOULD NOT retain and/or store the CLIENTID information WITH authentication credentials or authentication systems directly, but the IMAP service MAY associate the CLIENTID with a specific account holder, eg to create a history file of known CLIENTID tokens associated or permitted to access or present authentication credentials for that account holder.

This document recommends that an IMAP server handle any given client identity type from a CLIENTID command in one or more of the following manners.

1. Handled but treat as not presented (ignored, no persistence)
2. Store in IMAP session but treat as not presented (debugging)
3. Store in the IMAP session, so it is available to System log
4. Store in the IMAP session, so it is available to User log
5. Use for authentication
6. Use for alert when authentication fails
7. Use for alert when authentication succeeds
8. Unused

#### 7. Examples

##### 7.1. UUID as Client Identity

```
C: [connection established over a plaintext connection]
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS AUTH=GSSAPI LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 STARTTLS
S: a002 OK STARTTLS completed
<TLS negotiation, further commands are under [TLS] layer>
C: a003 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=GSSAPI AUTH=PLAIN CLIENTID
S: a003 OK CAPABILITY completed
C: a004 CLIENTID UUID 23bf83be-aad7-46aa-9e0f-39191ccf402f
S: a004 OK CLIENTID completed
C: a005 LOGIN joe password
S: a005 OK LOGIN completed
```



## 7.2. Malformed CLIENTID Command

```
C: [connection established over a plaintext connection]
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS AUTH=GSSAPI LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 STARTTLS
S: a002 OK STARTTLS completed
<TLS negotiation, further commands are under [TLS] layer>
C: a003 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=GSSAPI AUTH=PLAIN CLIENTID
S: a003 OK CAPABILITY completed
C: a004 CLIENTID UUID
S: a004 BAD Error in IMAP command received by server
```

The IMAP server rejects the CLIENTID command as it is not well formed due to there being only a single parameter provided.

## 7.3. Client Identity without TLS/SSL Session

```
C: [connection established over a plaintext connection]
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS AUTH=GSSAPI LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 CLIENTID UUID 23bf83be-aad7-46aa-9e0f-39191ccf402f
S: a002 BAD Unknown IMAP command received by server
```

The IMAP server rejects use of the CLIENTID command as the CLIENTID capability had not been advertised because no encryption was negotiated between the IMAP server and IMAP client.

## 7.4. Client Identity Leading to Rejection

```
C: [connection established over a plaintext connection]
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS AUTH=GSSAPI LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 STARTTLS
S: a002 OK STARTTLS completed
<TLS negotiation, further commands are under [TLS] layer>
C: a003 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=GSSAPI AUTH=PLAIN CLIENTID
S: a003 OK CAPABILITY completed
C: a004 CLIENTID UUID 23bf83be-aad7-46aa-9e0f-39191ccf402f
S: a004 OK CLIENTID completed
C: a005 LOGIN joe password
S: a005 BAD Failed to authenticate
```

The IMAP server rejects use of the system during the LOGIN command after deciding that the provided client identity does not establish sufficient privileges. Note that the error message that's returned to the client is very generic and does not reveal any information about CLIENTID and/or the existence of 'joe' and/or the validity of the password.



## 8. Security Considerations

As this extension provides an additional means of communicating information from a client to a server it is clear there is additional information divulged to the server. This may have privacy considerations depending on the client identity type or its contents. For example, it may reveal a MAC address of the device used to communicate with a server that would not previously have been revealed. While it has been useful to use identifier such as email address for authentication it is easy for these authentication tokens to be shared and/or reused and/or be publically available for other purposes. An IMAP server and or its operators SHOULD not share any CLIENTID information presented with a third party as it may represent or be linked to an individual and SHOULD never be shared in association with authentication tokens.

As well, while this service extension requires that the identity information only be transmitted over an encrypted channel to reduce the risk of eavesdropping, it does not specify any policies or practices required in the establishment of such a channel, and so it is the responsibility of the client and the server to determine that the communication medium meets their requirements.

## 9. IANA Considerations

The IANA is requested to add CLIENTID to the "IMAP 4 Capabilities" registry, <http://www.iana.org/assignments/imap4-capabilities>.

## 10. References

### 10.1. Normative References

- [ABNF] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.
- [IMAP] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.
- [KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [SASL] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.
- [TLS] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.



## Appendix A. CLIENTID Product Support

Since publishing the IMAP Client Identity RFC draft, multiple email server and client vendors have implemented CLIENTID support into their products, e.g. MailEnable, MagicMail, SaneBox, BlueMail, emClient, and Thunderbird.

## Contributors

Michael Peddemors  
LinuxMagic

## Authors' Addresses

Deion Yu  
LinuxMagic  
#405 - 860 Homer St.  
Vancouver, British Columbia  
CA V6B 2W5

EMail: [deionyu@linuxmagic.com](mailto:deionyu@linuxmagic.com)



