Network Working Group
Internet-Draft
Intended status: Informational
Expires: 2 March 2026

S. Gougeon

29 August 2025

The IMAP WEBPUSH extension
draft-gougeon-imap-webpush-03

Abstract

This document defines a WEBPUSH extension of the Internet Message Access Protocol (IMAP) that permits IMAP servers to send WebPush notifications.

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 2 March 2026.

Copyright Notice

Table of Contents

1. Introduction

WebPush (defined by [RFC8030], [RFC8291] and [RFC8292]) defines a way for applications to deliver real-time events in a timely fashion, with push notifications. Push notifications allows consolidating all real-time events into a single session which ensures more efficient use of network and radio resources. They are particularly used in mobile environments. Push notifications may also be used for other use cases, for example during the migration to a new server, the new

server may subscribe to the old server to be notified when an event is recorded.

Many use cases have led to a need for real-time events with email. IMAP support for real-time events has been added with the IDLE command ([RFC2177], [RFC9051]) and the NOTIFY extension ([RFC5465]). These commands require using a persistent connection per account and contribute to unnecessary use of the device radio.

JMAP ([RFC8620]) has responded to this need by supporting WebPush from the beginning.

Therefore, this extension permits IMAP servers to send WebPush notifications.

2. Conventions and Definitions

In examples, "C:" and "S:" indicate lines sent by the client and server, respectively. Lines ending in "\" are interrupted for presentation reasons, they would actually be joined to the next line. Note that each other line includes the terminating CRLF.

User agent, is defined in [RFC8030] as a device and software that is the recipient of push messages. It describes here the mail client.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Overview

This extension adds 5 commands: GETVAPID, WEBPUSH, ACKWEBPUSH, LWEBPUSH and SILWEBPUSH. GETVAPID allows to get the server VAPID public key, WEBPUSH to subscribe a new push subscription, ACKWEBPUSH to confirm the subscription information is correct, LWEBPUSH to list current subscriptions and SILWEBPUSH to silence a subscription for a given time.

Each time the IMAP server records a subscribed event, it sends a push message to the registered push endpoints. The notification overflow event enables IMAP servers to implement a generic push notification.

4. CAPABILITY Identification

IMAP servers that support this extension MUST include "WEBPUSHdraft1" in the CAPABILITY command response. This will be "WEBPUSH" once the extension is officially published.

5. Client Commands

5.1. GETVAPID Command

This command is only available in authenticated or selected state.

Arguments: none

Responses: REQUIRED untagged response: VAPID

Result: OK - capability completed

BAD - arguments invalid

The GETVAPID command requests the VAPID public key ([RFC8292]) of the server. the server MUST send a single untagged VAPID response before the tagged OK response. This is used by clients to request a push endpoint on their push server restricted to this mail server.

Example:

```
C: a1 GETVAPID
S: * VAPID \
    BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
    zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
S: a1 OK GETVAPID completed
```

5.2. WEBPUSH Command

This command is only available in authenticated or selected state.

Arguments (create/update): subscription ID

subscription name

push endpoint

ECDH public key

authentication secret

filter

Arguments (delete): subscription ID

NIL

Responses (inactive subscription): untagged response: WEBPUSH

VAPID

Responses (active subscription): OPTIONAL untagged response: WEBPUSH

VAPID

Result: OK - creation or deletion completed

NO - creation failure: can’t create webpush subscription with these arguments

BAD - command unknown or arguments invalid

The WEBPUSH command creates, updates or delete a push subscription for the account. New subscriptions and updated subscriptions (with a new endpoint, a new public key or a new authentication secret) aren’t active until the subscription is confirmed with the ACKWEBPUSH command.

The subscription ID is unique per account and identifies the push subscription. Sending a WEBPUSH command with an existing subscription ID updates the current subscription. Sending a WEBPUSH command with an existing subscription ID following by NIL deletes the subscription.

The subscription name is not unique per account, it may be used to know to which client a subscription belong.

If the client tries to delete a subscription with an unknown subscription ID, the server returns a tagged OK response.

If the subscription is inactive or is deactivated by the command, the server MUST return a single untagged WEBPUSH response before the tagged OK response and MUST return a single untagged VAPID response before the tagged OK response. The VAPID response can be used by the client to check if the server has rotated its VAPID keys.

If the subscription is still active after the command, the server MAY return a single untagged WEBPUSH response and MAY return a single untagged VAPID response before the tagged OK response.

If the subscription was activated (i.e. created and then confirmed/ acknowledged) with a VAPID key that is not the current VAPID key (due to a key rotation), then subscription MUST be deactivated by this command, and a confirm/acknowledgement is required to activate the subscription again.

If the updated or created subscription is inactive after the command (this is a new subscription, at least one field is different or the VAPID key has been rotated), the server MUST send a push notification with a AckSubscription event, encrypted with the new public key, to the new endpoint. The client will have to send a ACKWEBPUSH command to (re-)activate the subscription. The client knows if the subscription is deactivated thanks to the untagged WEBPUSH response.

The push endpoint MUST be the URI that the mail server sends push messages to. This is defined as the URI for push resource in [RFC8030]. This URI MUST use the "https" scheme.

The ECDH public key is the user agent public key on the P-256 curve. It MUST be encoded in the uncompressed form [SEC_1] (section 2.3.3, replicated from X9.62), and base64url encoded as described in [RFC7515]. This is used to encrypt push notifications following [RFC8291].

The authentication secret is 16 random bytes. It MUST be base64url encoded as described in [RFC7515]. This is used to encrypt push notifications following [RFC8291].

The filter follows the syntax defined in [RFC5465]. It contains the mailboxes to be watched and the events about which to notify the client. Supported events are listed in the dedicated section (Section 7). SELECTED and SELECTED-DELAYED mailbox specifier can be used indifferently, they both refer to the selected mailbox without any difference. Supported attributes for new messages may be specified as follow the NOTIFY, SEARCH and SORT commands.

The client SHOULD reregister their push subscription from time to time, like every time the client starts, in order to restore subscriptions, in case the endpoint was removed. Subscription may be removed if the push server has been in an inconsistent state, or if the mail server has been restored from a backup.

The capabilities that were enabled using the ENABLE command at the time of the WEBPUSH command are associated with the subscription. When sending push messages, the untagged messages are formatted according to those capabilities. For example, enabling IMAP4rev2 or UTF8=ACCEPT cause mailboxes to be in UTF-8, and enabling CONDSTORE causes MODSEQ to be added to some events.

SEARCH and SORTS command sent after a WEBPUSH request do not modify any push subscription.

Example:

```
C: a1 LWEBPUSH *
S: a1 OK LWEBPUSH completed
C: a2 WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client \
    https://push.example.net/push/random1 \
    BCVxsr7N_eNgVRqvHtD0zTZsEc6-VV-JvLexhqUzORcxaOzi6-AYWXvTBHm4bj\
    yPjs7Vd8pZGH6SRpkNtoIAiw4 \
    BTBZMqHH6r4Tts7J_aSIgg \
    (personal (Messagenew messageExpunge))
S: * VAPID BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
    zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client NIL
S: a2 OK WEBPUSH completed
C: a3 WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client \
    https://push.example.net/push/JzLQ3raZJfFBR0aqvOMsLrt54w4rJUsV \
    BCVxsr7N_eNgVRqvHtD0zTZsEc6-VV-JvLexhqUzORcxaOzi6-AYWXvTBHm4bj\
    yPjs7Vd8pZGH6SRpkNtoIAiw4 \
    BTBZMqHH6r4Tts7J_aSIgg \
    (personal (Messagenew messageExpunge))
S: * VAPID BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
    zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client NIL
S: a3 OK WEBPUSH completed
C: a4 LWEBPUSH *
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client NIL
S: a4 OK LWEBPUSH completed
C: a5 WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 NIL
S: a5 OK WEBPUSH completed
C: a6 LWEBPUSH *
S: a6 OK LWEBPUSH completed
```

5.3. ACKWEBPUSH Command

This command is only available in authenticated or selected state.

Arguments: acknowledgement token, sendi by the server with a push message containing an AckSubscription event

Responses: untagged response: WEBPUSH

Result: OK - The subscription is activated

NO - The token doesn’t exist or is expired

BAD - arguments invalid

Example to activate a subscription with a valid token:

```
C: a1 LWEBPUSH *
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client NIL
S: a1 OK LWEBPUSH completed
C: a2 ACKWEBPUSH 585078c5-fb8b-4ed0-8e77-474ab08f0a30
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 0
S: a2 OK ACKWEBPUSH completed
```

Example with an unknown or expired token:

```
C: a1 ACKWEBPUSH 5aa04cf0-f156-406e-84af-3cee534b23b8
S: a1 NO ACKWEBPUSH completed
```

5.4. LWEBPUSH Command

This command is only available in authenticated or selected state.

Arguments: subscription ID with possible wildcards

Responses: untagged response: WEBPUSH

Result: OK - list completed

NO - list failure: can’t list webpush records

BAD - arguments invalid

The LWEBPUSH command returns a subset of webpush subscriptions from the complete set of all subscriptions available to the client. Zero or more untagged WEBPUSH responses are returned, containing information to identified the subscriptions. The server MUST return the WEBPUSH response for the exact subscription ID if the account has a subscription with this ID. It MUST return all the account’s subscriptions if the argument is a wildcard "*".

Example:

```
C: a1 LWEBPUSH *
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 0
    https://push.endpoint.tld/random1 0
S: * WEBPUSH 80a3b492-bc9c-46a9-91ab-5866b27073bb \
    my-mobile-client NIL
    https://push.endpoint.tld/random2 NIL
S: * WEBPUSH 28626e4e-37d1-456c-a667-5258b5528508 \
    my-desktop-client 1112
S: a1 OK LWEBPUSH completed
C: a2 LWEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 0
S: a2 OK LWEBPUSH completed
```

5.5. SILWEBPUSH Command

This command is only available in authenticated or selected state.

Arguments: subscription ID

duration to silence subscription, in seconds. MUST be equal or higher than 0

OR "session" to silence the subscription as long as the connection is alive.

Responses: OPTIONAL untagged response: WEBPUSH

Result: OK - create completed

NO - the subscription is unknown or inactive

BAD - command unknown or arguments invalid

The SILWEBPUSH command allows the client to silence a push subscription for a duration. It may be useful for clients synchronizing with the server when they receive a push notification, without parsing it. It may also be useful for clients that support a "do not disturb" period. When the duration is 0, the push subscription is active again and the server will send events for this subscription.

Clients MAY announce their subscription ID with SILWEBPUSH subscription-id session, which causes the server to not send any push notification as long as the connection is alive. This isn’t reflected in WEBPUSH responses. A server MAY refuse to silent for the session more than one subscription from the same connection.

For example, when a client receives a push notification, it may silence the subscription for the duration (or a little less) that it stays connected to the server:

Example:

```
C: a1 SILWEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 10
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 10
S: a1 OK SILWEBPUSH completed
C: a2 SILWEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 session
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 10
S: a2 OK SILWEBPUSH completed
```

6. Server Responses

6.1. VAPID Response

The VAPID response occurs as a result of a GETVAPID command. It returns the server VAPID public key ([RFC8292]). This is a public key on the P-256 curve. It MUST be encoded in the uncompressed form [SEC_1] (section 2.3.3, replicated from X9.62), and base64url encoded as described in [RFC7515]. The server MAY use a different key pair for each account.

Example:

```
S: * VAPID BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
    zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
```

6.2. WEBPUSH Response

The WEBPUSH response occurs as a result of a LWEBPUSH command. It MAY be returned as a result of a WEBPUSH, or a SILWEBPUSH command too. It contains the subscription ID, the subscription name, and if the subscription is active, the silenced duration in seconds, else nil.

Example:

```
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 0
```

7. Supported events

The WEBPUSH command enables getting MessageNew, MessageExpunge, AnnotationChange, or FlagChange notifications for the subscribed mailboxes. After a new subscription, the server sends a AckSubscription notification to enable the subscription.

7.1. AckSubscription

A push notification containing a AckSubscription event is sent by the server when a webpush subscription is created or updated with the WEBPUSH command. The JSON contains the following fields:

* eventType: string, "AckSubscription"

* token: string, a random token

The random token SHOULD be a UUIDv4 token. The token SHOULD be valid for 10 minutes.

The token is then send to the server with the ACKWEBPUSH command in order to activate the new or updated subscription.

Example:

```
{
    "eventType": "AckSubscription",
    "token": "1d93db43-0b71-42d1-abfd-7baaa740208b"
}
```

7.2. FlagChange and AnnotationChange

FlagChange covers the MessageRead, MessageTrash, FlagsSet, and FlagsClear events in [RFC5423].

When the flag and/or message annotation change happens, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "FlagChange" or "AnnotationChange"

* mailbox: string, the mailbox concerned by the event

* uid: int, the IMAP UID of the concerned message

* flags: array of strings, containing the new message flags

If CONDSTORE [RFC4551] and/or QRESYNC [RFC5162] are enabled by the client before sending the WEBPUSH command, then the push notification also contains:

* highestmodseq int, The highest mod-sequence value of all messages in the mailbox

* uidvalidity int, the UIDVALIDITY of the mailbox

If the number of messages with the \Seen flag changes, the server MAY also include the UNSEEN data item in the push notification:

* unseen int, the number of messages with the \Seen flag

Example:

```
{
    "eventType": "FlagChange",
    "mailbox": "My Mailbox",
    "uid": 9999,
    "flags": ["$Junk"],
    "highestmodseq": 65535,
    "uidvalidity": 101
}
```

7.3. MessageNew

This covers both MessageNew and MessageAppend in [RFC5423].

When a new/appended message arrives in a subscribed mailbox, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "MessageNew"

* mailbox: string, the mailbox concerned by the event

* uid: int, the IMAP UID of the new or appended message

Depending on the subscription filter, the push notifications contain the following message attributes. If the subscription filter don’t any attribute, then all fields MUST be in the JSON:

* from: array, if the filter contains body.peek[header.fields (from)] an array of JSON containing:

  - name: optional string, the name of the origin

  - email: string, the email address of the origin

* to: array, if the filter contains body.peek[header.fields (to)] an array of JSON containing:

  - name: optional string, the name of the destination

  - email: string, the email address of the destination

* date: string, the received date if the filter contains body.peek[header.fields (date)]

* subject: string, the subject of the email if the filter contains body.peek[header.fields (subject)]

The push notification SHOULD contain the content of the message, if the final payload fits in the push notification size limit (4096 bytes once encrypted):

* content: string, the body of the message, encoded with type of contentType

* contentType: string, indicate the original media type of a resource before any content encoding is applied, REQUIRED IF content is present

* contentEncoding: string, encoding type of the content, REQUIRED IF content is present

If the push notification doesn’t contain the content of the message because it doesn’t fit in the size limit, then the server MAY add: - preview: string, preview of the message in UTF-8 plaintext, SHOULD be at least 256 bytes long.

If FlagChange events are also included in the subscription filter, the JSON MUST contain:

* flags: array of strings, containing the new message flags

If CONDSTORE [RFC4551] and/or QRESYNC [RFC5162] are enabled by the client before sending the WEBPUSH command, then the push notification also contains:

* highestmodseq int, The highest mod-sequence value of all messages in the mailbox

* uidvalidity int, the UIDVALIDITY of the mailbox

Example:


```
{
    "eventType": "MessageNew",
    "mailbox": "My Mailbox",
    "uid": 1001,
    "from": [{"email": "cdup@domain.tld", "name": "Camille"}]
    "to": [{"email": "me@example.tld"}],
    "date": "2025-04-09T13:40:39Z",
    "subject": "Hello",
    "content": "--------------MIME-BOUNDARY------------\nContent-Type: application/octet-
stream; name=\"encrypted.asc\"[...]",
    "contentType": "text/plain",
    "contentEncoding": "multipart/encrypted;protocol=\"application/pgp-encrypted\";bounda
ry=\"--------------MIME-BOUNDARY------------\""
}
```


7.4. MessageExpunge

If MessageExpunge is enabled, and a message is expunged, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "MessageExpunge"

* mailbox: string, the mailbox concerned by the event

* uid: int, the IMAP UID of the expunged message

If CONDSTORE [RFC4551] and/or QRESYNC [RFC5162] are enabled by the client before sending the WEBPUSH command, then the push notification also contains:

* highestmodseq int, The highest mod-sequence value of all messages in the mailbox

* uidvalidity int, the UIDVALIDITY of the mailbox

Example:


```
{
    "eventType": "MessageExpunge",
    "mailbox": "My Mailbox",
    "uid": 1001
}
```


7.5. MailboxName

If MailboxName is enabled, the server sends push notifications when a mailbox is created, deleted or renamed.

When a mailbox is created, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "MailboxCreated"

* mailbox: string, the new mailbox name

When a mailbox is deleted, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "MailboxDeleted"

* mailbox: string, the deleted mailbox name

When a mailbox is renamed, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "MailboxDeleted"

* oldName: string, the old mailbox name

* newName: string, the new mailbox name

Example of a mailbox created:

```
{
    "eventType": "MailboxCreated",
    "mailbox": "My Mailbox"
}
```

Example of a mailbox renamed:

```
{
    "eventType": "MailboxRenamed",
    "oldName": "My Mailbox",
    "newName": "My_Mailbox"
}
```

7.6. SubscriptionChange

If SubscriptionChange is enabled, when the subscription changes, for example when a mailbox is added to the personal collection, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "SubscriptionChange"

* mailboxes: array of string, containing all the subscribed mailbox names

Example of a mailbox renamed:

```
{
    "eventType": "SubscriptionChange",
    "mailboxes": ["INBOX", "My Mailbox"]
}
```

7.7. MailboxMetadataChange

Support for this event type is OPTIONAL unless the METADATA extension [RFC5464] is also supported by the server, in which case support for this event type is REQUIRED.

If enabled, when the metadata of a mailbox changed, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "MailboxMetadataChange"

* mailbox: string, the mailbox concerned by the event

* entries: array of string, list of entries, each of which have changed on the mailbox

Example of a mailbox renamed:

```
{
    "eventType": "MailboxMetadataChange",
    "mailbox": "INBOX",
    "entries": ["/shared/comment", "/private/comment"]
}
```

7.8. ServerMetadataChange

Support for this event type is OPTIONAL unless the METADATA or the METADATA-SERVER extension [RFC5464] is also supported by the server, in which case support for this event type is REQUIRED.

If enabled, when the metadata of the server changed, the server sends a push notification containing a JSON with the following fields:

* eventType: string, "ServerMetadataChange"

* entries: array of string, list of entries, each of which have changed on the server

Example of a mailbox renamed:

```
{
    "eventType": "ServerMetadataChange",
    "entries": ["/shared/comment", "/private/comment"]
}
```

7.9. Notification overflow

Instead of sending the full content for an event, for instance if the content of a push event is too long, the server MAY send an overflow event.

This event is a JSON with the following fields:

* eventType: string, "Overflow"

* forEventType, optional string, the event type of the overflowed event. The server can send a generic overflow event to inform the client should synchronize with the server by not sending this field.

* mailboxes, optional array of string, this is the mailboxes concerned by the event. This field is RECOMMENDED if the event is related to mailboxes.

Example:

```
{
    "eventType": "Overflow",
    "forEventType": "MessageNew",
    "mailboxes": ["My Mailbox"]
}
```

8. Push notifications

Once an account has one or more push subscription (registered with WEBPUSH command), the server sends a push message for each subscription every time a subscribed event is recorded.

The notification is encrypted following [RFC8291] specifications, send following [RFC8030] and authorized with [RFC8292].

8.1. Content

The server can send push notifications at any time. The push notification consist of a JSON with: - pushId: u32, a counter for the push subscription, increased by one every time a push is sent. The client may wait a few seconds if it receives a push notification with an higher id than expected, and synchronize with the server if it doesn't receive it. If the client receives a push notification with a lower id than expected, the client MUST store the id as the last id. The server MAY use that behavior to reset the push counter. events: array, an array of push events, as desribed in the supported events section.

As stated in RFC8291, the cleartext content of push notifications MUST NOT be longer than 3993 bytes. If the server wants to inform the client about a response longer than that, it MAY send a SYNC message.

Example of a push notification containing a new message for the "My Mailbox" mailbox, and one deleted message for the "INBOX" mailbox:


```
{
    "pushId": 1001,
    "events": [
        {
            "eventType": "MessageNew",
            "mailbox": "My Mailbox",
            "uid": 1001,
            "from": [{"email": "cdup@domain.tld", "name": "Camille"}]
            "to": [{"email": "me@example.tld"}],
            "date": "2025-04-09T13:40:39Z",
            "subject": "Hello",
            "content": "--------------MIME-BOUNDARY------------\nContent-Type: applicatio
n/octet-stream; name=\"encrypted.asc\"[...]",
            "contentType": "text/plain",
            "contentEncoding": "multipart/encrypted;protocol=\"application/pgp-encrypted\
";boundary=\"--------------MIME-BOUNDARY------------\""
        },
        {
            "eventType": "MessageExpunge",
            "mailbox": "INBOX",
            "uid": 1000
        }
    ]
}
```


Example of a push notification containing overflowed events:

```
{
    "pushId": 1002,
    "events": [
        {
            "eventType": "Overflow",
            "forEventType": "MessageNew",
            "mailboxes": ["My Mailbox"],
        },
        {
            "eventType": "Overflow",
            "forEventType": "MessageExpunge",
            "mailboxes": ["INBOX"],
        }
    ]
}
```

Example of a generic push notification, to implement synchronization on push:

```
{
    "pushId": 1003,
    "events": [
        {
            "eventType": "Overflow"
        }
    ]
}
```

8.2. WebPush headers

Push messages SHOULD have 604800 for the TTL header (a week).

Push messages with a MessageNew event SHOULD have high for the Urgency header, other push messages SHOULD have normalfor the Urgency header.

Push messages SHOULD NOT contain a Topic header.

8.3. Push message processing

Because all events regarding a specific mailbox, sent by push notifications contains a mailbox field, the client MAY parse that field only and choose to synchronize with the server. For example, when the client receives "mailbox": "INBOX", it requests the server for new events.

The client MAY also parse only some other events. For example, a client may parse MessageNew events, so it can directly show important informations on the user interface. And synchronize with the server if it receives another response.

The client MUST NOT rely exclusively on push notifications to stay synchronized as they may arrive out of order, and delivery isn't guarenteed. Especially if the client uses the contents of the push messages, or if the client performs partial synchronization on push (synchronize a single mailbox), the client MUST rely on another mechanism to be fully synchronized. For example, a periodic synchronization may be used.

8.4. Push server response

When the push server returns a 429 Too many requests, it should have send a Retry-After headers [RFC9110] to indicate how long the server has to wait before sending another request. If this header is present, the server MUST follow the period requested. If this header is not present, the mail server SHOULD wait 5 minutes before sending another request to this endpoint.

When the push server returns another 4XX status code, the mail server MUST removes the subscription.

When the push server returns a 5XX status code, the mail server SHOULD wait 5 minutes before sending another request to the endpoint.

9. VAPID key rotation

Server VAPID key rotation may be necessary in some cases.

When the server rotates the keys, if it still has access to the old VAPID key, the server SHOULD send the new VAPID public key with a VAPID response in a push notification, using the old VAPID key for the authorization.

The server MAY keep using the old VAPID key for some time. As soon as the old VAPID key is invalidated, old subscriptions activated with this VAPID key MUST be deactivated.

When the server receives a WEBPUSH command for an activated subscription that has been acknowledged with the old VAPID key, the server MUST deactivate the subscription and MUST reply with an untagged VAPID response and an untagged WEBPUSH response. The server then MUST send an AckSubscription event with a new token in a push notification.

All pending acknowledgement tokens must be invalidate when the VAPID key is rotated.

Example of a client sending a WEBPUSH command for a subscription activated with the old VAPID key:

```
C: a1 LWEBPUSH *
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client 0
S: a1 OK LWEBPUSH completed
C: a2 WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client \
    https://push.example.net/push/random1 \
    BCVxsr7N_eNgVRqvHtD0zTZsEc6-VV-JvLexhqUzORcxaOzi6-AYWXvTBHm4bj\
    yPjs7Vd8pZGH6SRpkNtoIAiw4 \
    BTBZMqHH6r4Tts7J_aSIgg
S: * VAPID BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
    zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
    my-mobile-client NIL
    https://push.example.net/push/random1 NIL
S: a2 OK WEBPUSH completed
```

10. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234].

Non-terminals referenced but not defined below are as defined by [RFC9051] and [RFC4466]. event-groups is defined in [RFC5465].

```
capability =/ "WEBPUSH"

command-auth =/ getvapid / webpush / ackwebpush / lwebpush /
silwebpush

getvapid = "GETVAPID"

webpush = "WEBPUSH" SP (add-webpush-opt / del-webpush-opt)

ackwebpush = "ACKWEBPUSH" SP ackwebpush-token

lwebpush = "LWEBPUSH" SP (list-wildcards / subscription-id)

silwebpush = "SILWEBPUSH" SP subscription-id SP (duration /
"session")
```

```
add-webpush-opt = subscription-id SP subscription-name SP endpoint SP
pubkey SP auth SP event-groups ; event-groups is defined in RFC5465

del-webpush-opt = subscription-id SP nil

response-payload =/ vapid-resp / webpush-resp / ackwebpush-resp /
sync-resp

vapid-resp = "VAPID" SP pubkey

webpush-resp = "WEBPUSH" SP subscription-id SP subscription-name SP
(duration / nil)

sync-resp = "SYNC" [SP response-first-word]

push-content = 1*(response-data)

response-first-word = <response-payload until first SP>

subscription-id = atom ; Case sensitive

subscription-name = atom

endpoint = "https://" text ; Case sensitive

pubkey = 87(base64url-char)

auth = 22(base64url-char)

duration = number

base64url-char = ALPHA / DIGIT / "_" / "-" ; Case sensitive
```

11. Security Considerations

The privacy and security considerations of [RFC8030] [RFC8291] and [RFC8292] all apply to the use of this extension.

WebPush on decentralized applications may be used as a DDOS amplification, by registering multiple time a target as the endpoint, on multiple servers then notifying all the accounts. Requiring the client to acknowledge one push notification (with the ACKWEBPUSH command and response) greatly reduces this risk.

12. IANA Considerations

Registration of a new IMAP capability in the IMAP Capability registry requires the publication of a standards- track RFC or an IESG approved experimental RFC. The registry is currently located at http://www.iana.org/assignments/imap4-capabilities (http://www.iana.org/assignments/imap4-capabilities). This document defines the WEBPUSH IMAP capability need to be registered to the registry.

13. References

13.1. Normative References

[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>.

[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>.

[RFC5464] Daboo, C., "The IMAP METADATA Extension", RFC 5464, DOI 10.17487/RFC5464, February 2009, <https://www.rfc-editor.org/info/rfc5464>.

[RFC5465] Gulbrandsen, A., King, C., and A. Melnikov, "The IMAP NOTIFY Extension", RFC 5465, DOI 10.17487/RFC5465, February 2009, <https://www.rfc-editor.org/info/rfc5465>.

[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>.

[RFC8030] Thomson, M., Damaggio, E., and B. Raymor, Ed., "Generic Event Delivery Using HTTP Push", RFC 8030, DOI 10.17487/RFC8030, December 2016, <https://www.rfc-editor.org/info/rfc8030>.

[RFC8292] Thomson, M. and P. Beverloo, "Voluntary Application Server Identification (VAPID) for Web Push", RFC 8292, DOI 10.17487/RFC8292, November 2017, <https://www.rfc-editor.org/info/rfc8292>.

[RFC8291] Thomson, M., "Message Encryption for Web Push", RFC 8291, DOI 10.17487/RFC8291, November 2017, <https://www.rfc-editor.org/info/rfc8291>.

[RFC9051] Melnikov, A., Ed. and B. Leiba, Ed., "Internet Message Access Protocol (IMAP) - Version 4rev2", RFC 9051, DOI 10.17487/RFC9051, August 2021, <https://www.rfc-editor.org/info/rfc9051>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9586] Melnikov, A., Achuthan, A. P., Nagulakonda, V., Singh, A., and L. Alves, "IMAP Extension for Using and Returning Unique Identifiers (UIDs) Only", RFC 9586, DOI 10.17487/RFC9586, May 2024, <https://www.rfc-editor.org/info/rfc9586>.

[SEC_1] "SEC 1: Elliptic Curve Cryptography", n.d., <https://www.secg.org/sec1-v2.pdf>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[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>.

13.2. Informative References

[RFC2177] Leiba, B., "IMAP4 IDLE command", RFC 2177, DOI 10.17487/RFC2177, June 1997, <https://www.rfc-editor.org/info/rfc2177>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

Author’s Address

Simon Gougeon
Email: ietf@sgougeon.fr