JMAP
Internet-Draft
Intended status: Standards Track
Expires: December 18, 2020

R. Ouazana, Ed.
Linagora

June 16, 2020

# Handling Message Disposition Notification with JMAP
# draft-ietf-jmap-mdn-10

Abstract

JMAP ([RFC8620] - JSON Meta Application Protocol) is a generic protocol for synchronising data, such as mail, calendars or contacts, between a client and a server. It is optimised for mobile and web environments, and aims to provide a consistent interface to different data types.

JMAP for Mail ([RFC8621] - The JSON Meta Application Protocol (JMAP) for Mail) specifies a data model for synchronising email data with a server using JMAP. Clients can use this to efficiently search, access, organise, and send messages.

MDN are defined in [RFC8098] and are used as "read receipts", "acknowledgements", or "receipt notifications".

MDN have a specific format that must be parsed or generated. The goal of this document is to specify a data model for handling MDN messages with a server using JMAP.

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 December 18, 2020.

Table of Contents



## 1. Introduction

JMAP ([RFC8620] - JSON Meta Application Protocol) is a generic protocol for synchronising data, such as mail, calendars or contacts, between a client and a server. It is optimised for mobile and web environments, and aims to provide a consistent interface to different data types.

JMAP for Mail ([RFC8621] - The JSON Meta Application Protocol (JMAP) for Mail) specifies a data model for synchronising email data with a

server using JMAP. Clients can use this to efficiently search, access, organise, and send messages.

MDN are defined in [RFC8098] and are used as "read receipts", "acknowledgements", or "receipt notifications".

A client can have to deal with MDN in different ways:

1. When receiving an email, an MDN can be sent to the sender. This specification defines an MDN/send method to cover this case.

2. When sending an email, an MDN can be requested. This must be done with the help of a header, and is already specified by [RFC8098] and can already be handled by [RFC8621] this way.

3. When receiving an MDN, the MDN could be related to an existing sent mail. This is already covered by [RFC8621] in the EmailSubmission object. Client might want to display detailed information about a received MDN. This specification defines an MDN/parse method to cover this case.

## 1.1. Notational conventions

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.

Type signatures, examples and property descriptions in this document follow the conventions established in section 1.1 of [RFC8620]. Data types defined in the core specification are also used in this document.

Servers MUST support all properties specified for the new data types defined in this document.

## 1.2. Terminology

The same terminology is used in this document as in the core JMAP specification.

Keywords being case insensitive in IMAP but JSON being case sensitive, the "$mdnsent" keyword MUST always be used in lowercase.

### 1.3. Addition to the capabilities object

Capabilities are announced as part of the standard JMAP Session resource; see [RFC8620], section 2.

The capability "urn:ietf:params:jmap:mdn" being present in the "accountCapabilities" property of an account represents support for the "MDN" data type, parsing MDN via the "MDN/parse" method, and creating and sending MDN messages via the "MDN/send" method. Servers that include the capability in one or more "accountCapabilities" properties MUST also include the property in the "capabilities" property.

The value of this "urn:ietf:params:jmap:mdn" property is an empty object in the account's "accountCapabilities" property.

## 2. MDN

An *MDN* object has the following properties:

- forEmailId: "Id|null" Email Id of the received email this MDN is relative to. This property MUST NOT be null for "MDN/send", but may be null in the response from the "MDN/parse" method.

- subject: "String|null" Subject used as "Subject" header for this MDN.

- textBody: "String|null" Human readable part of the MDN, as plain text.

- includeOriginalMessage: "Boolean" (default: false). If "true", the content of the original message will appear in the third component of the multipart/report generated for the MDN. See [RFC8098] for details and security considerations.

- reportingUA: "String|null" Name of the MUA creating this MDN. It is used to build the MDN Report part of the MDN.

- disposition: "Disposition" Object containing the diverse MDN disposition options.

- mdnGateway: "String|null" (server-set) Name of the gateway or MTA that translated a foreign (non-Internet) message disposition notification into this MDN.

- originalRecipient: "String|null" (server-set) Original recipient address as specified by the sender of the message for which the MDN is being issued.

- finalRecipient: "String|null" Recipient for which the MDN is being issued. if set, it overrides the value that would be calculated by the server from the Identity.

- originalMessageId: "String|null" (server-set) Message-ID (the [RFC5322] header field, not the JMAP Id) of the message for which the MDN is being issued.

- error: "String[]|null" (server-set) Additional information in the form of text messages when the "error" disposition modifier appears.

- extensionFields: "String[String]|null" (server-set) Object where keys are extension-field names and values are extension-field values.

A *Disposition* object has the following properties:

- actionMode: "String" This MUST be one of the following strings: "manual-action" / "automatic-action"

- sendingMode: "String" This MUST be one of the following strings: "mdn-sent-manually" / "mdn-sent-automatically"

- type: "String" This MUST be one of the following strings: "deleted" / "dispatched" / "displayed" / "processed"

See [RFC8098] for the exact meaning of these different fields. These fields are defined case insensitive in [RFC8098] but are case sensitive in this RFC and MUST be converted to lowercase by "MDN/parse".

## 2.1. MDN/send

The MDN/send method sends an [RFC5322] message from an MDN object. When calling this method the "using" property of the Request object MUST contain the capabilities "urn:ietf:params:jmap:mdn" and "urn:ietf:params:jmap:mail". The latter because of the implicit call to Email/set and the use of Identities, described below. The method takes the following arguments:

- accountId: "Id" The id of the account to use.

- identityId: "Id" The id of the Identity to associate with these MDN. The server will use this identity to define the sender of the MDN and to set the finalRecipient field.

- send: "Id[MDN]" A map of creation id (client specified) to MDN objects.

- onSuccessUpdateEmail: "Id[PatchObject]|null" A map of creation id to an object containing properties to update on the Email object referenced by the "MDN/send" if the sending succeeds.

The response has the following arguments:

- accountId: "Id" The id of the account used for the call.

- sent: "Id[MDN]|null" A map of creation id to MDN containing any properties that were not set by the client. This includes any properties that were omitted by the client and thus set to a default by the server. This argument is null if no MDN objects were successfully sent.

- notSent: "Id[SetError]|null" A map of the creation id to a SetError object for each record that failed to be sent, or null if all successful.

The following already registered SetError would mean:

- notFound: The reference Email Id cannot be found, or has no valid "Disposition-Notification-To" header.

- forbidden: MDN/send would violate an ACL or other permissions policy.

- forbiddenFrom: The user is not allowed to use the given finalRecipient property.

- overQuota: MDN/send would exceed a server-defined limit on the number or total size of sent MDN. It could include limitations on sent emails.

- tooLarge: MDN/send would result in an MDN that exceeds a server-defined limit for the maximum size of an MDN, or more generally on emails.

- rateLimit: Too many MDN or emails have been created recently, and a server-defined rate limit has been reached. It may work if tried again later.

- invalidProperties: The record given is invalid in some way.

The following is a new SetError:

- mdnAlreadySent: The message has the "$mdnsent" keyword already set.

If the accountId or identityId given cannot be found, the method call is rejected with an "invalidArguments" error.

The client SHOULD NOT issue an MDN/send request if the message has the "$mdnsent" keyword set.

When sending the MDN, the server is in charge of generating the "originalRecipient", "finalRecipient" and "originalMessageId" fields according to the [RFC8098] specification.

The client is expected to explicitly update each "Email" for which an "MDN/send" has been invoked in order to set the "$mdnsent" keyword on these emails. To ensure that, the server MUST reject an "MDN/send" which does not result in setting the keyword "$mdnsent". Thus the server MUST check that the "onSuccessUpdateEmail" property of the method is correctly set to update this keyword.

## 2.2. MDN/parse

This method allows a client to parse blobs as [RFC5322] messages to get MDN objects. This can be used to parse and get detailed information about blobs referenced in the "mdnBlobIds" of the EmailSubmission object, or any email the client could expect to be an MDN.

The "forEmailId" property can be null or missing if the "originalMessageId" property is missing, not referencing an existing email or if the server cannot efficiently calculate the related email (for example if several emails get the same "Message-Id" header).

The MDN/parse method takes the following arguments:

- accountId: "Id" The id of the account to use.

- blobIds: "Id[]" The ids of the blobs to parse.

The response has the following arguments:

- accountId: "Id" The id of the account used for the call.

- parsed: "Id[MDN]|null" A map of blob id to parsed MDN representation for each successfully parsed blob, or null if none.

- notParsable: "Id[]|null" A list of ids given that corresponded to blobs that could not be parsed as MDNs, or null if none.

- notFound: "Id[]|null" A list of blob ids given that could not be found, or null if none.

The following additional errors may be returned instead of the MDN/parse response:

- requestTooLarge: The number of ids requested by the client exceeds the maximum number the server is willing to process in a single method call.

- invalidArguments: If the accountId given cannot be found, the MDN parsing is rejected with an "invalidArguments" error.

## 3. Samples

### 3.1. Sending an MDN for a received email

A client can use the following request to send an MDN back to the sender:

```
[[ "MDN/send", {
  "accountId": "ue150411c",
  "identityId": "I64588216",
  "send": {
    "k1546": {
      "forEmailId": "Md45b47b4877521042cec0938",
      "subject": "Read receipt for: World domination",
      "textBody": "This receipt shows that the email has been
          displayed on your recipient's computer. There is no
          guaranty it has been read or understood.",
      "reportingUA": "linagora.com; OpenPaaS",
      "disposition": {
        "actionMode": "manual-action",
        "sendingMode": "mdn-sent-manually",
        "type": "displayed"
      }
    }
  },
  "onSuccessUpdateEmail": {
    "#k1546": {
      "keywords/$mdnsent": true
    }
  }
}, "0" ]]
```

If the email id matches an existing email without the "$mdnsent" keyword, the server can answer:

```
    [[ "MDN/send", {
      "accountId": "ue150411c",
      "sent": {
        "k1546": {
          "finalRecipient": "rfc822; john@example.com",
          "originalMessageId": "<1521557867.2614.0.camel@apache.org>"
        }
      }
    }, "0" ],
    [ "Email/set", {
      "accountId": "ue150411c",
      "oldState": "23",
      "newState": "42",
      "updated": {
        "Md45b47b4877521042cec0938": {
          "keywords": {
            "$mdnsent": true
          }
        }
      }
    }, "0" ]]
```

If the "$mdnsent" keyword has already been set, the server can answer an error:

```
     [[ "MDN/send", {
       "accountId": "ue150411c",
       "notSent": {
         "k1546": {
           "type": "mdnAlreadySent",
           "description" : "$mdnsent keyword is already present"
         }
       }
     }, "0" ]]
```

## 3.2. Asking for MDN when sending an email

This is done with the [RFC8621] "Email/set" "create" method.

```
[[ "Email/set", {
  "accountId": "ue150411c",
  "create": {
    "k1546": {
      "mailboxIds": {
        "2ea1ca41b38e": true
      },
      "keywords": {
        "$seen": true,
        "$draft": true
      },
      "from": [{
        "name": "Joe Bloggs",
        "email": "joe@example.com"
      }],
      "to": [{
        "name": "John",
        "email": "john@example.com"
      }],
      "header:Disposition-Notification-To:asText": "joe@example.com",
      "subject": "World domination",
      ...
    }
  }
}, "0" ]]
```

Note the specified "Disposition-Notification-To" header indicating where to send MDN back (usually the sender of the email).

### 3.3. Parsing a received MDN

The client issues a parse request:

```
[[ "MDN/parse", {
  "accountId": "ue150411c",
  "blobIds: [ "0f9f65ab-dc7b-4146-850f-6e4881093965" ]
}, "0" ]]
```

The server responds:

```
    [[ "MDN/parse", {
      "accountId": "ue150411c",
      "parsed": {
        "0f9f65ab-dc7b-4146-850f-6e4881093965": {
          "forEmailId": "Md45b47b4877521042cec0938",
          "subject": "Read receipt for: World domination",
          "textBody": "This receipt shows that the email has been
              displayed on your recipient's computer. There is no
              guaranty it has been read or understood.",
          "reportingUA": "linagora.com; OpenPaaS",
          "disposition": {
            "actionMode": "manual-action",
            "sendingMode": "mdn-sent-manually",
            "type": "displayed"
          }
          "finalRecipient": "rfc822; john@example.com",
          "originalMessageId": "<1521557867.2614.0.camel@apache.org>"
        }
      }
    }, "0" ]]
```

In case of a not found blobId, the server would respond:

```
      [[ "MDN/parse", {
        "accountId": "ue150411c",
        "notFound": [ "0f9f65ab-dc7b-4146-850f-6e4881093965" ]
      }, "0" ]]
```

If the blobId has been found but is not parsable, the server would respond:

```
     [[ "MDN/parse", {
       "accountId": "ue150411c",
       "notParsable": [ "0f9f65ab-dc7b-4146-850f-6e4881093965" ]
     }, "0" ]]
```

## 4. IANA Considerations

### 4.1. JMAP Capability Registration for "mdn"

IANA will register the "mdn" JMAP Capability as follows:

Capability Name: "urn:ietf:params:jmap:mdn"

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, section 5.

### 4.2. JMAP Error Codes Registry

The following subsection register one new error code in the "JMAP Error Codes" registry, as defined in [RFC8620].

#### 4.2.1. mdnAlreadySent

JMAP Error Code: mdnAlreadySent

Intended use: common

Change controller: IETF

Reference: This document, Section 2.1

Description: The message has the "$mdnsent" keyword already set. The client MUST NOT try again to send an MDN for this message.

## 5. Security considerations

The same considerations regarding MDN (see [RFC8098] and [RFC3503]) apply to this document.

In order to enforce trust regarding the relation between the user sending an email and the identity of this user, the server SHOULD validate in conformance to the provided Identity that the user is permitted to use the finalRecipient value and return a forbiddenFrom error if not.

## 6. References

### 6.1. 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>.

[RFC3503] Melnikov, A., "Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP)", RFC 3503, DOI 10.17487/RFC3503, March 2003, <https://www.rfc-editor.org/info/rfc3503>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC8098] Hansen, T., Ed. and A. Melnikov, Ed., "Message Disposition Notification", STD 85, RFC 8098, DOI 10.17487/RFC8098, February 2017, <https://www.rfc-editor.org/info/rfc8098>.

[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>.

[RFC8621] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP) for Mail", RFC 8621, DOI 10.17487/RFC8621, August 2019, <https://www.rfc-editor.org/info/rfc8621>.

### 6.2. Informative References

[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>.

## Author's Address

Raphael Ouazana (editor)
Linagora
100 Terrasse Boieldieu - Tour Franklin
Paris - La Defense CEDEX 92042
France

Email: rouazana@linagora.com
URI: https://www.linagora.com