Workgroup: JMAP
Internet-Draft: draft-ietf-jmap-blob-00
Updates: 8620 (if approved)
Published: 12 July 2021
Intended Status: Standards Track
Expires: 13 January 2022
Authors: B. Gondwana, Ed.
Fastmail

JMAP Blob management extension

Abstract

The JMAP base protocol (RFC8620) provides the ability to upload and download arbitrary binary data via HTTP PUT and GET on defined endpoint. This binary data is called a "Blob".

This extension adds additional ways to create and access Blobs, by making inline method calls within a standard JMAP request.

This extension also adds a reverse lookup mechanism to discover where blobs are referenced within other datatypes.

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 13 January 2022.

Table of Contents

1. Introduction

Sometimes JMAP ([RFC8620]) interactions require creating a Blob and then referencing it. In the same way that IMAP Literals ([RFC7888]) were extended to reduce roundtrips for simple data, embedding simple small blobs into the JMAP method stream can reduce roundtrips.

Likewise, when fetching an object, it can be useful to also fetch the raw content of that object without a separate roundtrip.

Where JMAP is being proxied through a system which applies additional access restrictions, it can be useful to be able to see where a blob is referenced in order to decide whether to allow it to be downloaded.

2. Conventions Used In This Document

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. Blobs

A blob is a sequence of zero or more octets.

The JMAP base spec [RFC8210] defines the Blob/copy method, which is unchanged by this specfication.

3.1. Blob/set

This is a standard JMAP set method.

3.1.1. create

Properties:

Any one of:

- *data:asText: String|null
- *data:asBase64: String|null
- *data:asHex: String|null
- *catenate: [SetObject] *list of octet sources in order*

Also:

- *type: String|null

Result is:

- *id: Id the blobId
- *type: String|null *as given in the creation (if any); or detected from content; or null*
- *size: UnsignedInt *as per RFC8620 - the size of the file in Octets*

Any other properties identical to those that would be returned in the JSON response of the RFC8620 upload endpoint.

SetObject:

Any one of

- *data:asText: String|null
- *data:asBase64: String|null
- *data:asHex: String|null

OR a blobId source:

*blobId: Id

*offset: UnsignedInt|null

*length: UnsignedInt|null

3.1.2. update

It is not possible to update a Blob, so any update will result in a notUpdated response.

3.1.3. destroy

If an uploaded Blob is not referenced by any persistent object, the server SHOULD destroy the object. Some systems use a content-based ID for blobs, so the server MAY respond destroyed and yet that blobId still exist with the same content.

Example:

Method Call:

```
[ "Blob/set", {
  "accountId" : "account1",
  "create" : {
    "1": {
      "data:asBase64": "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABAQMAAAAl21bKA
                        AAAA1BMVEX/AAAZ4gk3AAAAAXRSTlN/gFy0ywAAAApJRE
                        FUeJxjYgAAAAYAAzY3fKgAAAAASUVORK5CYII=",
      "type" : "image/png"
    },
  },
}, "R1" ]
```

Response:

```
[ "Blob/set", {
  "accountId" : "account1",
  "created" : {
    "1": {
      "id" : "G4c6751edf9dd6903ff54b792e432fba781271beb",
      "type" : "image/png",
      "size" : 95
    },
  },
}, "R1" ]
```

3.2. Blob/get

A standard JMAP get.

Properties:

Any of

*data:asText

*data:asBase64

*data:asHex

*data *selects data:asText if the content is UTF-8, or data:asBase64*

*size

If not given, returns data and size.

QUESTION: do we want to add range operators?

*offset: UnsignedInt|null

*length: UnsignedInt|null

Returns that range of octets (not characters!) from the blob.
Alternative possible syntax - ranges within the properties, e.g:
data:asText:0:3000

3.3. Blob/lookup

Given a list of blobIds, this method does a reverse lookup in each of the provided datatypes to find the list of Ids within that datatype which reference the provided blob.

The definition of reference is somewhat loosely defined, but roughly means "you could discover this blobId by looking inside this object", for example if a Mailbox contains an Email which references the blobId, then it references that blobId. Likewise for a Thread.

Parameters

*accountId: "Id"

The id of the account used for the call.

*datatypes: [String]

A list of datatype names from the "JMAP Datatypes" registry for which "Can Reference Blobs" is "Yes". The capability which defines each type must also be requested.

If a datatype is not known by the server or the associated capability has not been included then the server returns an "unknownDataType" error.

*ids: [Id]

A list of blobId values to be looked for.

Response

*list: [BlobInfo]

A list of BlobInfo objects.

BlobInfo Object

*id: Id

The Blob Identifier.

*datatypes: DataType[Id List]

A map from datatype to list of Ids of that datatype (e.g. the datatype "Email" maps to a list of emailIds)

e.g.

```
[ "Blob/lookup", {
  "datatypes": ["Mailbox", "Thread", "Email"],
  "ids": ["Gd2f81008cf07d2425418f7f02a3ca63a8bc82003",
          "G6f954bcb620f7f50fc8f21426bde3669da3d9067"]
}, "R1" ]
```

Response:

```
[ "Blob/lookup", {
  "list": [
    {
      "id": "Gd2f81008cf07d2425418f7f02a3ca63a8bc82003",
      "datatypes": {
        "Mailbox": ["M54e97373", Mcbe6b662"],
        "Thread": ["T1530616e"],
        "Email": ["E16e70a73eb4", "E84b0930cf16"]
      }
    }
  ],
  "notFound": ["G6f954bcb620f7f50fc8f21426bde3669da3d9067"]
}, "R1"]
```

4. Security considerations

TO BE IMPROVED:

JSON parsers are not all consistent in handling non-UTF-8 data. JMAP requires that all JSON data be UTF-8 encoded, so servers MUST either return data:asBase64 or isEncodingProblem: true and modify the data to be UTF-8 safe.

Servers MUST apply any access controls such that if the authenticated user would be unable to discover the blobId by making queries, then this fact can't be discovered via a Blob/lookup. For example, if an Email exists in a Mailbox which the authenticated user does not have access to see, then that emailId MUST not be returned in a lookup for a blob which is referenced by that email.

If a blob is not visible to a user at all, then the server SHOULD return that blobId in the notFound array, however it may also return an empty list for each datatype, as it may not be able to know if other datatypes do reference that blob.

5. IANA considerations

5.1. JMAP Capability registration for "blob"

IANA is requested to register the "blob" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:blob

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

5.2. JMAP Error Codes Registration for "unknownDataType"

IANA is requested to register the "unknownDataType" JMAP Error Code as follows:

JMAP Error Code: unknownDataType

Intended use: common

Change Controller: IETF

Reference: this document

Description: The server does not recognise this data type, or the capability to enable it was not present.

5.3. Creation of "JMAP Datatypes" Registry

IANA is requested to create a new registry "JMAP Datatypes" with the initial content:

Datatype name	Can Reference Blobs	Can use for State Change	Capability	Reference
Core	No	No	urn:ietf:params:jmap:core	[RFC8620]
PushSubscription	No	No	urn:ietf:params:jmap:core	[RFC8620]
Mailbox	Yes	Yes	urn:ietf:params:jmap:mail	[RFC8621]
Thread	Yes	Yes	urn:ietf:params:jmap:mail	[RFC8621]
Email	Yes	Yes	urn:ietf:params:jmap:mail	[RFC8621]
EmailDelivery	No	Yes	urn:ietf:params:jmap:mail	[RFC8621]
SearchSnippet	No	No	urn:ietf:params:jmap:mail	[RFC8621]
Identity	No	Yes	urn:ietf:params:jmap:submission	[RFC8621]
EmailSubmission	No	Yes	urn:ietf:params:jmap:submission	[RFC8621]
VacationResponse	No	Yes	urn:ietf:params:jmap:vacationresponse	[RFC8621]
MDN	No	No	urn:ietf:params:jmap:mdn	[RFC9007]

Table 1

6. Acknowledgements

TBD

7. 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>.

[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>.

[RFC8210] Bush, R. and R. Austein, "The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1", RFC 8210, DOI 10.17487/RFC8210, September 2017, <https://www.rfc-editor.org/info/rfc8210>.

8. Informative References

[RFC7888] Melnikov, A., Ed., "IMAP4 Non-synchronizing Literals", RFC 7888, DOI 10.17487/RFC7888, May 2016, <https://www.rfc-editor.org/info/rfc7888>.

[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>.

Author's Address

Bron Gondwana (editor)
Fastmail
Level 2, 114 William St
Melbourne VIC 3000
Australia

Email: brong@fastmailteam.com
URI: https://www.fastmail.com