Internet Engineering Task Force (IETF)
Request for Comments: 9404
Updates: 8620
Category: Standards Track
ISSN: 2070-1721

B. Gondwana, Ed.
Fastmail

August 2023

# JSON Meta Application Protocol (JMAP) Blob Management Extension

## Abstract

The JSON Meta Application Protocol (JMAP) base protocol (RFC 8620) provides the ability to upload and download arbitrary binary data via HTTP POST and GET on a 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 data types.

## Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9404.



## Table of Contents

## 1. Introduction

Sometimes JMAP [RFC8620] interactions require creating a blob and then referencing it. In the same way that IMAP literals were extended by [RFC7888], embedding small blobs directly into the JMAP method calls array can be an option for reducing round trips.

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

Since raw blobs may contain arbitrary binary data, this document defines a use of the base64 coding specified in [RFC4648] for both creating and fetching blob data.

When JMAP is proxied through a system that applies additional access restrictions, it can be useful to know which objects reference any particular blob; this document defines a way to discover those references.

## 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.

The definitions of JSON keys and datatypes in the document follow the conventions described in [RFC8620].

## 3. Addition to the Capabilities Object

The capabilities object is returned as part of the JMAP Session object; see [RFC8620], Section 2.

This document defines an additional capability URI.

### 3.1. urn:ietf:params:jmap:blob

The presence of the capability urn:ietf:params:jmap:blob in the accountCapabilities property of an account represents support for additional API methods on the Blob datatype. Servers that include the capability in one or more accountCapabilities properties MUST also include the property in the capabilities property.

The value of this property in the JMAP Session capabilities property MUST be an empty object.

The value of this property in an account's accountCapabilities property is an object that MUST contain the following information on server capabilities and permissions for that account:

* maxSizeBlobSet: "UnsignedInt|null"

  The maximum size of the blob (in octets) that the server will allow to be created (including blobs created by concatenating multiple data sources together).

  Clients MUST NOT attempt to create blobs larger than this size.

  If this value is null, then clients are not required to limit the size of the blob they try to create, though servers can always reject creation of blobs regardless of size, e.g., due to lack of disk space or per-user rate limits.

* maxDataSources: "UnsignedInt"

  The maximum number of DataSourceObjects allowed per creation in a Blob/upload.

  Servers MUST allow at least 64 DataSourceObjects per creation.

* supportedTypeNames: "String[]"

  An array of data type names that are supported for Blob/lookup. If the server does not support lookups, then this will be the empty list.

  Note that the supportedTypeNames list may include private types that are not in the "JMAP Data Types" registry defined by this document. Clients MUST ignore type names they do not recognise.

* supportedDigestAlgorithms: "String[]"

  An array of supported digest algorithms that are supported for Blob/get. If the server does not support calculating blob digests, then this will be the empty list. Algorithms in this list MUST be present in the "HTTP Digest Algorithm Values" registry defined by [RFC3230]; however, in JMAP, they must be lowercased, e.g., "md5" rather than "MD5".

Clients SHOULD prefer algorithms listed earlier in this list.

### 3.1.1. Capability Example

```
{
  "capabilities": {
    ...,
    "urn:ietf:params:jmap:blob": {}
  },
  "accounts": {
    "A13842": {
      ...
      "accountCapabilities": {
        "urn:ietf:params:jmap:blob": {
          "maxSizeBlobSet": 50000000,
          "maxDataSources": 100,
          "supportedTypeNames" : [
            "Mailbox",
            "Thread",
            "Email"
          ],
          "supportedDigestAlgorithms" : [
            "sha",
            "sha-256"
          ]
        }
      }
    }
  }
}
```

## 4. Blob Methods

A blob is a sequence of zero or more octets.

JMAP [RFC8620] defines the Blob/copy method, which is unchanged by this specification and is selected by the urn:ietf:params:jmap:core capability.

The following JMAP methods are selected by the urn:ietf:params:jmap:blob capability.

### 4.1. Blob/upload

This is similar to a Foo/set in [RFC8620] in some ways. However, blobs cannot be updated or deleted, so only create is allowed in the method call. Also, blobs do not have state, so there is no state field present in the method response.

*Parameters*

* accountId: "Id"

  The id of the account in which the blobs will be created.

*  create: "Id[UploadObject]"

   A map of creation id to UploadObjects.

*Result*

The result is the same as for Foo/set in [RFC8620], with created and notCreated objects mapping from the creation id.

The created objects contain:

*  id: "Id"

   The blobId that was created.

*  type: "String|null"

   The media type as given in the creation (if any).  If not provided, the server MAY perform content analysis and return one of the following: the calculated value, "application/octet-string", or null.

*  size: "UnsignedInt"

   As per [RFC8620], the size of the created blob in octets.

The created objects will also contain any other properties identical to those that would be returned in the JSON response of the upload endpoint described in [RFC8620].  This may be extended in the future; in this document, it is anticipated that implementations will extend both the upload endpoint and the Blob/upload responses in the same way.

If there is a problem with a creation, then the server will return a notCreated response with a map from the failed creation id to a SetError object.

For each successful upload, servers MUST add an entry to the createdIds map ([RFC8620], Section 3.3) for the request; even if the caller did not explicitly pass a createdIds, the value must be available to later methods defined in the same Request Object.  This allows the blobId to be used via back-reference in subsequent method calls.

The created blob will have the same lifetime and same expiry semantics as any other binary object created via the mechanism specified in [RFC8620], Section 6.

Uploads using this mechanism will be restricted by the maxUploadSize limit for JMAP requests specified by the server, and clients SHOULD consider using the upload mechanism defined by [RFC8620] for blobs larger than a megabyte.

*UploadObject*

*  data: "DataSourceObject[]"

An array of zero or more octet sources in order (zero to create an empty blob). The result of each of these sources is concatenated in order to create the blob.

* type: "String|null" (default: null)

A hint for media type of the data.

*DataSourceObject*

Exactly one of:

* data:asText: "String|null" (raw octets, must be UTF-8)

* data:asBase64: "String|null" (base64 representation of octets)

or a blobId source:

* blobId: "Id"

* offset: "UnsignedInt|null" (MAY be zero)

* length: "UnsignedInt|null" (MAY be zero)

If null, then offset is assumed to be zero.

If null, then length is the remaining octets in the blob.

If the range cannot be fully satisfied (i.e., it begins or extends past the end of the data in the blob), then the DataSourceObject is invalid and results in a notCreated response for this creation id.

If the data properties have any invalid references or invalid data contained in them, the server MUST NOT guess the user's intent and MUST reject the creation and return a notCreated response for that creation id.

Likewise, invalid characters in the base64 of data:asBase64 or invalid UTF-8 in data:asText MUST result in a notCreated response.

It is envisaged that the definition for DataSourceObject might be extended in the future, for example, to fetch external content.

A server MUST accept at least 64 DataSourceObjects per create, as described in Section 3.1 of this document.

#### 4.1.1. Blob/upload Simple Example

The data:asBase64 field is set over multiple lines for ease of publication here; however, the entire data:asBase64 field would be sent as a continuous string with no wrapping on the wire.

Method Call:

[

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

Response:

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

#### 4.1.2. Blob/upload Complex Example

Method Calls:

```
[
 [
  "Blob/upload",
  {
   "create": {
    "b4": {
     "data": [
      {
       "data:asText": "The quick brown fox jumped over the lazy dog."
      }
     ]
    }
   }
  },
  "S4"
 ],
```

```
[
  "Blob/upload",
  {
   "create": {
     "cat": {
       "data": [
         {
           "data:asText": "How"
         },
         {
           "blobId": "#b4",
           "length": 7,
           "offset": 3
         },
         {
           "data:asText": "was t"
         },
         {
           "blobId": "#b4",
           "length": 1,
           "offset": 1
         },
         {
           "data:asBase64": "YXQ/"
         }
       ]
     }
   }
  },
  "CAT"
],
[
  "Blob/get",
  {
   "properties": [
     "data:asText",
     "size"
   ],
   "ids": [
     "#cat"
   ]
  },
  "G4"
 ]
]
```

**Responses:**

```
[
  [
    "Blob/upload",
    {
      "oldState": null,
      "created": {
        "b4": {
          "id": "Gc0854fb9fb03c41cce3802cb0d220529e6eef94e",
```

```
            "size": 45,
            "type": "application/octet-stream"
          }
        },
        "notCreated": null,
        "accountId": "account1"
      },
      "S4"
    ],
    [
      "Blob/upload",
      {
        "oldState": null,
        "created": {
          "cat": {
            "id": "Gcc60576f036321ae6e8037ffc56bdee589bd3e23",
            "size": 19,
            "type": "application/octet-stream"
          }
        },
        "notCreated": null,
        "accountId": "account1"
      },
      "CAT"
    ],
    [
      "Blob/get",
      {
        "list": [
          {
            "id": "Gcc60576f036321ae6e8037ffc56bdee589bd3e23",
            "data:asText": "How quick was that?",
            "size": 19
          }
        ],
        "notFound": [],
        "accountId": "account1"
      },
      "G4"
    ]
  ]
```

### 4.2. Blob/get

A standard JMAP get, with two additional optional parameters:

* offset: "UnsignedInt|null"

  Start this many octets into the blob data. If null or unspecified, this defaults to zero.

* length: "UnsignedInt|null"

  Return at most this many octets of the blob data. If null or unspecified, then all remaining octets in the blob are returned. This can be considered equivalent to an infinitely large length

value, except that the isTruncated warning is not given unless the start offset is past the end of the blob.

*Request Properties:*

Any of:

* data:asText

* data:asBase64

* data (returns data:asText if the selected octets are valid UTF-8 or data:asBase64)

* digest:<algorithm> (where <algorithm> is one of the named algorithms in the supportedDigestAlgorithms capability)

* size

If not given, the properties default to data and size.

*Result Properties:*

* data:asText: "String|null"

  The raw octets of the selected range if they are valid UTF-8; otherwise, null.

* data:asBase64: "String"

  The base64 encoding of the octets in the selected range.

* digest:<algorithm>: "String"

  The base64 encoding of the digest of the octets in the selected range, calculated using the named algorithm.

* isEncodingProblem: "Boolean" (default: false)

* isTruncated: "Boolean" (default: false)

* size: "UnsignedInt"

  The number of octets in the entire blob.

The size value MUST always be the number of octets in the underlying blob, regardless of offset and length.

The data fields contain a representation of the octets within the selected range that are present in the blob. If the octets selected are not valid UTF-8 (including truncating in the middle of a multi-octet sequence) and data or data:asText was requested, then the key isEncodingProblem MUST be set to true, and the data:asText response value MUST be null. In the case where data was requested and the data is not valid UTF-8, then data:asBase64 MUST be returned.

If the selected range requests data outside the blob (i.e., the offset+length is larger than the blob), then the result is either just the octets from the offset to the end of the blob or an empty string if the offset is past the end of the blob. Either way, the isTruncated property in the result MUST be set to true to tell the client that the requested range could not be fully satisfied. If digest was requested, any digest is calculated on the octets that would be returned for a data field.

Servers SHOULD store the size for blobs in a format that is efficient to read, and clients SHOULD limit their request to just the size parameter if that is all they need, as fetching blob content could be significantly more expensive and slower for the server.

#### 4.2.1. Blob/get Simple Example

In this example, a blob containing the string "The quick brown fox jumped over the lazy dog." has blobId Gc0854fb9fb03c41cce3802cb0d220529e6eef94e.

The first method call requests just the size for multiple blobs, and the second requests both the size and a short range of the data for one of the blobs.

Method Calls:

```
[
  [
    "Blob/get",
    {
      "accountId" : "account1",
      "ids" : [
        "Gc0854fb9fb03c41cce3802cb0d220529e6eef94e",
        "not-a-blob"
      ],
      "properties" : [
        "data:asText",
        "digest:sha",
        "size"
      ]
    },
    "R1"
  ],
  [
    "Blob/get",
    {
      "accountId" : "account1",
      "ids" : [
        "Gc0854fb9fb03c41cce3802cb0d220529e6eef94e"
      ],
      "properties" : [
        "data:asText",
        "digest:sha",
        "digest:sha-256",
        "size"
      ],
```

```
      "offset" : 4,
      "length" : 9
    },
    "R2"
  ]
]
```

Responses:

```
[
 [
  "Blob/get",
  {
   "accountId": "account1",
   "list": [
    {
     "id": "Gc0854fb9fb03c41cce3802cb0d220529e6eef94e",
     "data:asText": "The quick brown fox jumped over the lazy dog.",
     "digest:sha": "wIVPufsDxBzOOALLDSIFKebu+U4=",
     "size": 45
    }
  ],
  "notFound": [
   "not-a-blob"
  ]
 },
 "R1"
],
[
 "Blob/get",
 {
  "accountId": "account1",
  "list": [
   {
    "id": "Gc0854fb9fb03c41cce3802cb0d220529e6eef94e",
    "data:asText": "quick bro",
    "digest:sha": "QiRAPtfyX8K6tm1iOAtZ87Xj3Ww=",
    "digest:sha-256": "gdg9INW7lwHK6OQ9u0dwDz2ZY/gubi0En0xlFpKt0OA=",
    "size": 45
   }
  ]
 },
 "R2"
]
]
```

### 4.2.2. Blob/get Example with Range and Encoding Errors

The b1 value is the text "The quick brown fox jumped over the \x81\x81 dog.", which contains an invalid UTF-8 sequence.

The results have the following properties:

* G1: Defaults to data and size, so b1 returns isEncodingProblem and a base64 value.

*  G2: Since data:asText was explicitly selected, does not attempt to return a value for the data, just isEncodingProblem for b1.

*  G3: Since only data:asBase64 was requested, there is no encoding problem, and both values are returned.

*  G4: Since the requested range could be satisfied as text, both blobs are returned as data:asText, and there is no encoding problem.

*  G5: Both blobs cannot satisfy the requested range, so isTruncated is true for both.

> Note: Some values have been wrapped for line length.  There would be no wrapping in the data:asBase64 values on the wire.

Method Calls:


```
[
  [
    "Blob/upload",
    {
      "create": {
        "b1": {
          "data": [
            {
              "data:asBase64": "VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wZW
                                Qgb3ZlciB0aGUggYEgZG9nLg=="
            }
          ]
        },
        "b2": {
          "data": [
            {
              "data:asText": "hello world"
            }
          ],
          "type" : "text/plain"
        }
      }
    },
    "S1"
  ],
  [
    "Blob/get",
    {
      "ids": [
        "#b1",
        "#b2"
      ]
    },
    "G1"
  ],
  [
    "Blob/get",
    {
```

```
      "ids": [
        "#b1",
        "#b2"
      ],
      "properties": [
        "data:asText",
        "size"
      ]
    },
    "G2"
  ],
  [
    "Blob/get",
    {
      "ids": [
        "#b1",
        "#b2"
      ],
      "properties": [
        "data:asBase64",
        "size"
      ]
    },
    "G3"
  ],
  [
    "Blob/get",
    {
      "offset": 0,
      "length": 5,
      "ids": [
        "#b1",
        "#b2"
      ]
    },
    "G4"
  ],
  [
    "Blob/get",
    {
      "offset": 20,
      "length": 100,
      "ids": [
        "#b1",
        "#b2"
      ]
    },
    "G5"
  ]
]
```

**Responses:**

```
[
  [
    "Blob/upload",
```

```
  {
    "oldState": null,
    "created": {
      "b2": {
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
        "size": 11,
        "type": "application/octet-stream"
      },
      "b1": {
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
        "size": 43,
        "type": "text/plain"
      }
    },
    "updated": null,
    "destroyed": null,
    "notCreated": null,
    "notUpdated": null,
    "notDestroyed": null,
    "accountId": "account1"
  },
  "S1"
],
[
  "Blob/get",
  {
    "list": [
      {
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
        "isEncodingProblem": true,
        "data:asBase64": "VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wZW
                          Qgb3ZlciB0aGUgYEgZG9nLg==",
        "size": 43
      },
      {
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
        "data:asText": "hello world",
        "size": 11
      }
    ],
    "notFound": [],
    "accountId": "account1"
  },
  "G1"
],
[
  "Blob/get",
  {
    "list": [
      {
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
        "isEncodingProblem": true,
        "size": 43
      },
      {
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
```

```
        "data:asText": "hello world",
        "size": 11
      }
    ],
    "notFound": [],
    "accountId": "account1"
  },
  "G2"
],
[
  "Blob/get",
  {
    "list": [
      {
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
        "data:asBase64": "VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wZW
                         Qgb3ZlciB0aGUgYEgZG9nLg==",
        "size": 43
      },
      {
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
        "data:asBase64": "aGVsbG8gd29ybGQ=",
        "size": 11
      }
    ],
    "notFound": [],
    "accountId": "account1"
  },
  "G3"
],
[
  "Blob/get",
  {
    "list": [
      {
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
        "data:asText": "The q",
        "size": 43
      },
      {
        "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
        "data:asText": "hello",
        "size": 11
      }
    ],
    "notFound": [],
    "accountId": "account1"
  },
  "G4"
],
[
  "Blob/get",
  {
    "list": [
      {
        "id": "G72cfa4804194563685d9a4b695f7ba20e7739576",
```

```
            "isTruncated": true,
            "isEncodingProblem": true,
            "data:asBase64": "anVtcGVkIG92ZXIgdGhlIIGBIGRvZy4=",
            "size": 43
          },
          {
            "id": "G2aae6c35c94fcfb415dbe95f408b9ce91ee846ed",
            "isTruncated": true,
            "data:asText": "",
            "size": 11
          }
        ],
        "notFound": [],
        "accountId": "account1"
      },
      "G5"
    ]
  ]
```

### 4.3. Blob/lookup

Given a list of blobIds, this method does a reverse lookup in each of the provided type names to find the list of Ids within that data type that reference the provided blob.

Since different datatypes will have different semantics of "contains", the definition of "reference" is somewhat loose but roughly means "you could discover this blobId by looking at this object or at other objects recursively contained within this object".

For example, with a server that supports [RFC8621], if a Mailbox references a blob and if any Emails within that Mailbox reference the blobId, then the Mailbox references that blobId. For any Thread that references an Email that references a blobId, it can be said that the Thread references the blobId.

However, this does not mean that if an Email references a Mailbox in its mailboxIds property, then any blobId referenced by other Emails in that Mailbox are also referenced by the initial Email.

*Parameters*

* accountId: "Id"

  The id of the account used for the call.

* typeNames: "String[]"

  A list of names from the "JMAP Data Types" registry or defined by private extensions that the client has requested. Only names for which "Can reference blobs" is true may be specified, and the capability that defines each type must also be used by the overall JMAP request in which this method is called.

  If a type name is not known by the server, or the associated capability has not been requested, 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 blobId.

* matchedIds: "String[Id[]]"

  A map from type name to a list of Ids of that data type (e.g., the name "Email" maps to a list of emailIds).

If a blob is not visible to a user or does not exist on the server at all, then the server MUST still return an empty array for each type as this doesn't leak any information about whether the blob is on the server but not visible to the requesting user.

#### 4.3.1. Blob/lookup Example

Method Call:

```
[
  "Blob/lookup",
  {
    "typeNames": [
      "Mailbox",
      "Thread",
      "Email"
    ],
    "ids": [
      "Gd2f81008cf07d2425418f7f02a3ca63a8bc82003",
      "not-a-blob"
    ]
  },
  "R1"
]
```

Response:

```
[
  "Blob/lookup",
  {
    "list": [
      {
        "id": "Gd2f81008cf07d2425418f7f02a3ca63a8bc82003",
```

```
        "matchedIds": {
          "Mailbox": [
            "M54e97373",
            "Mcbe6b662"
          ],
          "Thread": [
            "T1530616e"
          ],
          "Email": [
            "E16e70a73eb4",
            "E84b0930cf16"
          ]
        }
      }
    ],
    "notFound": [
      "not-a-blob"
    ]
  },
  "R1"
]
```

## 5. Security Considerations

All security considerations for JMAP [RFC8620] apply to this specification. Additional considerations specific to the data types and functionality introduced by this document are described here.

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 only return a null value if data:asText is requested for a range of octets that is not valid UTF-8 and set isEncodingProblem: true.

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 cannot be discovered via a Blob/lookup. For example, if an Email exists in a Mailbox that the authenticated user does not have access to see, then that emailId MUST NOT be returned in a lookup for a blob that is referenced by that email.

The server MUST NOT trust that the data given to a Blob/upload is a well-formed instance of the specified media type. Also, if the server attempts to parse the given blob, only hardened parsers designed to deal with arbitrary untrusted data should be used. The server SHOULD NOT reject data on the grounds that it is not a valid specimen of the stated type.

With carefully chosen data sources, Blob/upload can be used to recreate dangerous content on the far side of security scanners (anti-virus or exfiltration scanners, for example) that may be watching the upload endpoint. Server implementations SHOULD provide a hook to allow security scanners to check the resulting blob after concatenating the data sources in the same way that they do for the upload endpoint.

Digest algorithms can be expensive for servers to calculate. Servers

that share resources between multiple users should track resource usage by clients and rate-limit expensive operations to avoid resource starvation.

## 6. IANA Considerations

### 6.1. JMAP Capability Registration for "blob"

IANA has registered the "blob" JMAP capability as follows:

Capability Name: urn:ietf:params:jmap:blob
Specification document: RFC 9404
Intended use: common
Change Controller: IETF
Security and privacy considerations: RFC 9404, Section 5

### 6.2. JMAP Error Codes Registration for "unknownDataType"

IANA has registered the "unknownDataType" JMAP error code as follows:

JMAP Error Code: unknownDataType
Intended use: common
Change Controller: IETF
Reference: RFC 9404
Description: The server does not recognise this data type, or the capability to enable it is not present in the current Request Object.

### 6.3. Creation of "JMAP Data Types" Registry

IANA has created a new registry called "JMAP Data Types". Table 1 shows the initial contents of this new registry.

| Type Name | Can Ref 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

The registration policy for this registry is "Specification Required" [RFC8126]. Either an RFC or a similarly stable reference document defines a JMAP Data Type and associated capability.

IANA will appoint designated experts to review requests for additions to this registry, with guidance to allow any registration that provides a stable document describing the capability and control over the URI namespace to which the capability URI points.

# 7. References

## 7.1. 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>.

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/info/rfc3230>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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>.

[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>.


## 7.2. 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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC9007] Ouazana, R., Ed., "Handling Message Disposition Notification with the JSON Meta Application Protocol (JMAP)", RFC 9007, DOI 10.17487/RFC9007, March 2021, <https://www.rfc-editor.org/info/rfc9007>.

## Acknowledgements

Joris Baum, Jim Fenton, Neil Jenkins, Alexey Melnikov, Ken Murchison, Robert Stepanek, and the JMAP Working Group in the IETF.

## 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