Workgroup: JMAP
Internet-Draft:
draft-ietf-jmap-portability-guide-00
Published: 19 February 2024
Intended Status: Informational
Expires: 22 August 2024
Authors: J.M. Baum, Ed. audriga
H.J. Happel, Ed. audriga

JMAP Guide for Migration and Data Portability

Abstract

JMAP (RFC8620) is a generic, efficient, mobile friendly and scalable protocol that can be used for data of any type. This makes it a good fit for migrations or data portability use cases that are focusing on data import and export. However, due to its large set of features, it is also quite complex, which makes it difficult to explore new application domains in practice. The goal of this document is to provide guidelines on implementing essential parts of JMAP for a much lower entry barrier and more efficient implementation of the protocol.

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 22 August 2024.

Table of Contents

1. Introduction
  1.1. Conventions Used In This Document
2. Outline on RFC8620 Features
  2.1. Use cases
  2.2. Session Resource
  2.3. Structured Data exchange
  2.4. Standard Methods and Naming Convention
  2.5. Binary Data
3. Implementation Details
  3.1. Overview Tables
  3.2. Implementation considerations
    3.2.1. All use cases
    3.2.2. Listing
    3.2.3. Paging
    3.2.4. Dynamic Session Resource
    3.2.5. Attachments
  3.3. Recommended functionality
4. Security considerations
5. IANA Considerations
6. Acknowledgements
7. Normative References
Authors' Addresses

1. Introduction

JMAP [RFC8620] is designed to be a generic, efficient, mobile friendly and scalable protocol. This comes with the cost of high complexity, even though this is necessary to meet JMAP's design goals.

Migration and data portability is about moving arbitrary user data between services. JMAP is a particularly good fit for satisfying basic data portability requirements. It can be used as an open protocol in front of an application service, exposing data of any kind. However, implementing JMAP correctly can be complicated, which makes it difficult to explore new application domains in practice.

For basic data portability requirements, users need to be able to export their data from a product or import it into a product in real time. Providers that want to support JMAP for their service to meet data portability requirements likely do not want to implement the

full feature set that JMAP currently defines. Currently, there is no guidance on how to implement only parts of RFC8620's features.

This specification aims to provide guidance to identify essential parts of the JMAP spec for more rapid development. For the sole purpose of providing very basic data portability, there is no need to implement all parts of the JMAP protocol. In a second iteration developers could then extend upon this basic version of JMAP.

1.1. 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 the core JMAP specification [RFC8620].

2. Outline on RFC8620 Features

2.1. Use cases

Not all features of [RFC8620] are required for basic data portability use cases. This document focuses on three scenarios:

* JMAP Minimum: Minimal JMAP implementation possible to show what needs to be implemented at the very least. In practice, applications will choose to implement more than this bare minimum as this alone most likely does not satisfy any use case for JMAP.
* Export use cases: Minimal implementation for exporting data from an application server. A common scenario for data portability.
* Import use cases: Minimal implementation for importing data into an application server. A common scenario for data portability.

2.2. Session Resource

For a lot of basic portability use cases for existing application services the following constraints are typically acceptable:

* A single user login is tied to a single user account (this is then also the primary account).
* Shared data shall not be exposed over the API. The API will only provide access to data owned by the current user.
* Each user has the same set of capabilities and restrictions (e.g. maxMailboxesPerEmail).
* apiUrl, downloadUrl, uploadUrl and eventSourceUrl are the same for ever user.

For use cases adhering to those restrictions, the session resource can be modeled as a simple JSON file that only contains constant values. This JSON then specifies a single accountId which is then assigned the value "self".

2.3. Structured Data exchange

While batching improves performance considerably, it imposes additional implementation effort on developers. It is not essential for portability and can be left out.

2.4. Standard Methods and Naming Convention

JMAP core defines 6 standard methods. Not all JMAP Methods are required to provide essential portability. For some use-cases where the data is expected to be small, **/set** and **/get** should be enough. In case a large amount of data shall be supported, paging can be achieved via the **/query** method. Note that some specifications require specific IDs for /get .

/changes, **/copy** as well as **/queryChanges** are not required as all data can already be imported and exported with above's three methods.

2.5. Binary Data

The advanced Blob/copy method call is not essential. Not all applications support attachments for their specific kind of data. Additionally, some data types allow having attachments as base64 enconded strings inside a JMAP object. In those cases, it is not required to implement a download or upload endpoint.

3. Implementation Details

3.1. Overview Tables

Tables 1-4 list the required features for a minimal implementation of JMAP for Migration and Portability in more detail. The next section provides more implementation considerations. It references the three use cases defined in Section 2.1. The value for each table cell details what is required at minimum for an implementation:

* **-** : Not required
* **constant value(s)**: RFC 8620 requires that at least a constant value will be returned by an application
* **error response**: RFC 8620 requires that a minimal implementation emits an error response signalling that a feature is not supported
* **""**: Same as JMAP Minimum

JMAP Core Feature	JMAP Minium	JMAP Portability export use cases	JMAP Portability import use cases
Session Object	constant values[1]	""	""
Service Autodiscovery	-	""	""

Table 1: Session Object features essential for Migration and Portability use cases

JMAP Core Feature	JMAP Minimum	JMAP Portability export use cases	JMAP Portability import use cases
Invocation (all properties)	required	""	""
Request (using)	required	""	""
Request (methodCalls)	required	""	""
Request (createdIds)	-	""	""
Response (methodResponses)	required	""	""
Response (createdIds)	-	""	""
Response (sessionState)	constant value	""	""
Errors	required	""	""
References to Previous Method Results	-	""	""
Localisation of User-Visible String	-	""	""

Table 2: Structured Data Exchange features essential for Migration and Portability use cases

JMAP Core Feature	JMAP Minimum	JMAP Portability export use cases	JMAP Portability import use cases
Core/echo	required	""	""
/get method Request	error response	required	""
/get method Request (accountId)	-	constant value[1]	""
/get method Request (ids)	-	error response, required for listing/paging[2]	""
/get method Request (properties)	-	error response	""
/get method Response	-	required	""
/get method Response (accountId)	-	constant value[1]	""
	-	constant value	""

JMAP Core Feature	JMAP Minimum	JMAP Portability export use cases	JMAP Portability import use cases
/get method Response (state)			
/get method Response (list)	-	required	""
/get method Response (notFound)	-	constant value, required for listing/paging[2]	""
/changes method (full)	error response	""	""
/set method Request	error response	""	required
/set method Request (accountId)	-	""	constant value[1]
/set method Request (ifInState)	-	""	constant value
/set method Request (create, only single id)	-	""	required
/set method Request (create, multiple ids)	-	""	""
/set method Request (update)	-	""	error response
/set method Request (destroy)	-	""	error response
/set method Response	-	""	required
/set method Response (accountId)	-	""	constant value[1]
/set method Response (oldState)	-	""	constant value
/set method Response (newState)	-	""	constant value
/set method Response (created)	-	""	required
/set method Response (updated)	-	""	constant value
/set method Response (destroyed)	-	""	constant value
/set method Response (notCreated)	-	""	required
/set method Response (notUpdated)	-	""	error response
/set method Response (notDestroyed)	-	""	error response
/set method SetError	-	""	required
/copy method (full)		""	""

JMAP Core Feature	JMAP Minimum	JMAP Portability export use cases	JMAP Portability import use cases
	error response		
/query method Request	error response	required for listing/paging[2]	""
/query method Request (accountId)	-	constant value for listing / paging[1,2]	""
/query method Request (filter)	-	error response for listing / paging[2]	""
/query method Request (sort)	-	error response for listing / paging[2]	""
/query method Request (position)	-	error response for listing, required for paging[2]	""
/query method Request (anchor)	-	error response for listing / paging[2]	""
/query method Request (anchorOffset)	-	""	""
/query method Request (limit)	-	error response for listing / paging[2]	""
/query method Request (calculateTotal)	-	error response for listing, required for paging[2]	""
/query method Response	-	required for listing /paging[2]	""
/query method Response (accountId)	-	constant value for listing / paging[1,2]	""
/query method Response (queryState)	-	required for listing/paging[2]	""
/query method Response (canCalculateChanges)	-	constant value	""
/query method Response (position)	-	required for paging	""
/query method Response (ids)	-	required for listing/paging[2]	""
/query method Response (total)	-	required for paging	""

JMAP Core Feature	JMAP Minimum	JMAP Portability export use cases	JMAP Portability import use cases
/query method Response (limit)	-	required for listing/paging[2]	""
/query method FilterCondition	-	""	""
/query method FilterOperator	-	""	""
/query method Comparator	-	""	""
/queryChanges method (full)	error response	""	""

Table 3: Method features essential for Migration and Portability

JMAP Core Feature	JMAP Minimum	JMAP Portability export use cases	JMAP Portability import use cases
Uploading Binary Data	-	""	required for importing attachments[3]
Downloading Binary Data	-	required for exporting attachments[3]	""
Blob/copy (full)	-	""	""
Push	-	""	""

Table 4: Blob and Push features essential for Migration and Portability

*1: Can be modeled as a JSON file with only constant values for a lot of scenarios. Section 2.2 details for which use cases such a JSON file should be sufficient.

*2: maxObjectsInGet might not be sufficiently high in some products. In this case, /query is required. Similarly, paging might be required in case a query might exceed the limit for the maximum amount of results for /query.

*3: Some data types allow having attachments as base64 enconded strings inside a JMAP object. In those cases it is not necessary to implement a download or upload endpoint.

3.2. Implementation considerations

3.2.1. All use cases

3.2.1.1. Bare Minimum

For a bare minimum Session object, choose the following to return a static JSON, which is the same for every user:

*maxSizeRequest
*maxConcurrentRequests

*apiUrl
*additional capabilities and their configuration

```
{
  "capabilities": {
    "urn:ietf:params:jmap:core": {
      "maxSizeUpload": 0,
      "maxConcurrentUpload": 0,
      "maxSizeRequest": <maxSizeRequest>,
      "maxConcurrentRequests": <maxConcurrentRequests>,
      "maxCallsInRequest": 1,
      "maxObjectsInGet": 0,
      "maxObjectsInSet": 0,
      "collationAlgorithms": []
    },
    "urn:ietf:params:jmap:<other-capability>": {},
    ...
  },
  "accounts": {
    "self": {
      "name": "",
      "isPersonal": true,
      "isReadOnly": true,
      "accountCapabilities": {
        "urn:ietf:params:jmap:<other-capability>": {
          "<key>": <value>,
          ...
        },
        ...
      }
    }
  },
  "primaryAccounts": {
    "urn:ietf:params:jmap:<other-capability>": "self"
  },
  "username": "",
  "apiUrl": "<apiUrl>",
  "downloadUrl": "",
  "uploadUrl": "",
  "eventSourceUrl": "",
  "state": ""
}
```

Implement the following for Structured Data Exchange:

*Invocation (all properties) - **required**
*Request (using) - **required**; Make sure to keep capabilities simple to keep implementation complexity low.

  *Request (methodCalls) - **required**; No logic for batching, because maxCallsInRequest is set to 1.
  *Request (createdIds) - not required; Only required in conjunction with batching.
  *Response (methodResponses) - **required**
  *Response (createdIds) - not required; See createIds.
  *Response (sessionState) - *constant value*; Set to "".
  *Errors - **required**; Implement serverFail as the most basic of all error types. Make sure to implement its description property to clarify the error context. Additional error types are very useful for debugging.
  *References to Previous Method Results - not required; This is, because batching is not advertised.
  *Localisation of User-Visible String - not required

For Methods:

  *Core/echo method (full) - **required**; Core/echo is useful for basic connection testing and requires very low implementation effort.
  */get method (full) - *error response*; always reply with requestTooLarge error.
  */set method (full) - *error response*; always reply with accountReadOnly error.
  */changes method (full) - *error response*; always reply with cannotCalculateChanges error.
  */copy method (full) - *error response*; always reply with serverFail error. Its description should explain that the method is merely not supported.
  */query method (full) - *error response*; same as /copy.
  */queryChanges method (full) - *error response*; always reply with cannotCalculateChanges error.

Note that there are some caveats when implementing the bare minimum of JMAP. Setting downloadUrl and uploadUrl to an empty string might be incompatible with some existing JMAP implementations as they are defined as "MUST contain variables", which an empty string does not contain. Also, some errors recommended in this document, like serverFail or invalidArgument, typically signal to clients that something unexpected has happened, when in fact servers can expect clients to call any standard JMAP method or property. Due to this the description property is used to clarify the context of errors like serverFail and invalidArgument. However, its value is a free-text string and therefore not machine-processable in an interoperable way.

3.2.1.2. Exporting data

The Session Object now additional requires:

* maxObjectsInGet

For Methods:

* /get method Request - **required**
* /get method Request (accountId) - *constant value*; Is always "self". Can therefore be ignored on the server side.
* /get method Request (ids) - *error response*; Always respond with invalidArguments. Its description property MUST explain that the ids property is merely not supported. The ids property is formally required by RFC8920, but in practice applications will not use the ids fields without retrieving ids via /query or /set first.
* /get method Request (properties) - *error response*; Always respond with invalidArguments. Its description property MUST explain that the properties property is merely not supported.
* /get method Response
* /get method Response (accountId) - *constant value*; Always set to "self".
* /get method Response (state) - *constant value*; Always set to empty string.
* /get method Response (list) - **required**
* /get method Response (notFound) - *constant value*; Always set to empty array, see ids.

3.2.1.3. Importing data

The Session Object now additional requires:

* isReadOnly = false
* maxObjectsInSet

For methods:

* /set method Request - **required**
* /set method Request (accountId) - *constant value*; Is always "self". Can therefore be ignored on the server side.
* /set method Request (ifInState) - *constant value*; State is always empty string, so a simple check if this is empty string should suffice.
* /set method Request (create, only single id) - **required**
* /set method Request (create, multiple ids) - not required; Only required if maxObjectsInSet > 1.
* /set method Request (update) - *error response*; Always place in notUpdated and reply with forbidden SetError. Its description

property MUST explain that the update property is merely not supported.
*/set method Request (destroy) - *error response*; Always place in notDestroyed and reply with forbidden SetError. Its description property MUST explain that the update property is merely not supported.
*/set method Response - **required**
*/set method Response (accountId) - *constant value*; Is always "self". Can therefore be ignored on the server side.
*/set method Response (oldState) - *constant value*; Is always empty string.
*/set method Response (newState) - *constant value*; Is always empty string.
*/set method Response (created) - **required**
*/set method Response (updated) - *constant value*; Is always an empty array.
*/set method Response (destroyed) - *constant value*; Is always empty array.
*/set method Response (notCreated) - **required**
*/set method Response (notUpdated) - *error response*; In case update was not empty.
*/set method Response (notDestroyed) - *error response*; In case destroy was not empty.
*/set method SetError - **required**; Implement forbidden to signal update and destroy are not supported.

3.2.2. Listing

Methods:

*/get method Request (ids) - **required**
*/get method Response (notFound) - **required**
*/query method Request - **required**
*/query method Request (accountId) - *constant value*; Is always "self". Can therefore be ignored on the server side.
*/query method Request (filter) - *error response*; Always return unsupportedFilter.
*/query method Request (sort) - *error response*; Always return unsupportedSort.
*/query method Request (position) - *error response*; always return invalidArgument. Its description property MUST explain that the position property is merely not supported.
*/query method Request (anchor) - *error response*; always return invalidArgument. Its description property MUST explain that the anchor property is merely not supported.
*/query method Request (anchorOffset) - not required; This is because anchor is not supported.

*/query method Request (limit) - *error response*; Always return invalidArgument. Its description property MUST explain that the limit property is merely not supported.
*/query method Request (calculateTotal) - *error response*; Always return invalidArgument. Its description property MUST explain that the calculateTotal property is merely not supported.
*/query method Response - **required**
*/query method Response (accountId) - *constant value*; Is always "self". Can therefore be ignored on the server side.
*/query method Response (queryState) - **required**
*/query method Response (canCalculateChanges) - *constant value*; Is always false.
*/query method Response (position) - not required; position not supported
*/query method Response (ids) - **required**
*/query method Response (total) - not required
*/query method Response (limit) - **required**
*/query method FilterCondition - not required
*/query method FilterOperator - not required
*/query method Comparator - not required

3.2.3. Paging

Regarding methods, all requirements of Section 3.2.2, as well as:

*/query method Request (position) - **required**
*/query method Request (calculateTotal) - **required**
*/query method Response (position) - **required**
*/query method Response (total) - **required**

3.2.4. Dynamic Session Resource

3.2.4.1. Multi-Account Scenario

A user might have access to more than one account. In this case, the constraints defined in Section 2.2 no longer apply.

For the Session Object:

*choose how to determine accountId (e.g. base64-encode the username)
*make sure that accounts returns more than a single account if a user has more

Methods:

*method Request (accountId) - **required**; Can no loner be a constant value or ignored on the server side as a user might have multiple accounts.

*method Response (accountId) - **required**; Can no longer be the constant value "self".

3.2.4.2. Varying Capabilities or Server configuration between users

Each user might have different restrictions regarding account capabilities. In this case, accounts will may differ from one user to another, and the constraints defined in Section 2.2 no longer apply.

Other properties like uploadUrl, apiUrl etc. might change dynamically. It may no longer be possible to choose static values for them.

3.2.5. Attachments

3.2.5.1. Exporting attachments

For exporting data, choose for the Session Object:

*downloadUrl

Implement:

*actual download endpoint

3.2.5.2. Importing attachments

For importing data, choose for the Session Object:

*uploadUrl
*maxSizeUpload
*maxConcurrentUpload

Implement:

*actual upload endpoint

3.3. Recommended functionality

Autodiscovery is useful, so clients can use the endpoint more easily.

Destroying objects via /set is very valuable functionality for testing. Without it, JMAP cannot be used to remove data. It requires the following method implementation:

*/set method Request (destroy) - **required**
*/set method Response (destroyed) - **required**
*/set method Response (notDestroyed) - **required**

The filter functionality of /query may be relevant for your use case. Filters allow listing objects of a specific kind.

4. Security considerations

All security considerations of JMAP [RFC8620] apply to this specification.

5. IANA Considerations

This document has no IANA actions.

6. Acknowledgements

Bron Gondwana, Neil Jenkins, Alexey Melnikov, Ken Murchison, Robert Stepanek and the JMAP working group at the IETF.

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>.

[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>.

Authors' Addresses

Joris Baum (editor)
audriga
Alter Schlachthof 57
76137 Karlsruhe
Germany

Email: joris@audriga.com
URI: https://www.audriga.com

Hans-Joerg (editor)
audriga
Alter Schlachthof 57
76137 Karlsruhe
Germany

Email: hans-joerg@audriga.com

URI: https://www.audriga.com