Workgroup: JMAP
Internet-Draft: draft-ietf-jmap-contacts-00
Published: 17 October 2022
Intended Status: Standards Track
Expires: 20 April 2023
Authors: N.M. Jenkins, Ed.
Fastmail

JMAP for Contacts

Abstract

This document specifies a data model for synchronising contacts data 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 20 April 2023.

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.

This specification defines a data model for synchronising contacts between a client and a server using JMAP.

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.

1.2. Terminology

The same terminology is used in this document as in the core JMAP specification, see [RFC8620], Section 1.6.

The terms AddressBook, CardGroup, and Card (with these specific capitalizations) are used to refer to the data types defined in this document and instances of those data types.

1.3. Data Model Overview

An Account (see [RFC8620], Section 1.6.2) with support for the contacts data model contains zero or more AddressBook objects, which is a named collection of Cards and CardGroups. A Card is a representation of a person, company, or other entity in RFCXXXX JSContact Card format. A CardGroup is a named set of zero or more UIDs, used to represent a group of Cards, in RFCXXXX JSContact CardGroup format. Each Card or CardGroup belongs to exactly one AddressBook.

In servers with support for JMAP Sharing [RFC XXX], data may be shared with other users. Sharing permissions are managed per AddressBook.

1.4. Addition to the Capabilities Object

The capabilities object is returned as part of the JMAP Session object; see [RFC8620], Section 2. This document defines two additional capability URIs.

1.4.1. urn:ietf:params:jmap:contacts

This represents support for the AddressBook, CardGroup, and Card data types and associated API methods. The value of this property in the JMAP Session capabilities property is an empty object.

The value of this property in an account’ (U+2019)s accountCapabilities property is an object that MUST contain the following information on server capabilities and permissions for that account:

* **mayCreateAddressBook**: Boolean If true, the user may create an AddressBook in this account.

2. AddressBooks

An AddressBook is a named collection of Cards and CardGroups. All Cards and CardGroups are associated with exactly one AddressBook.

A **AddressBook** object has the following properties:

- ***id**: Id (immutable; server-set) The id of the AddressBook.

- ***name**: String The user-visible name of the AddressBook. This may be any UTF-8 string of at least 1 character in length and maximum 255 octets in size.

- ***isSubscribed**: Boolean Has the user indicated they wish to see this AddressBook in their client? This SHOULD default to false for AddressBooks in shared accounts the user has access to and true for any new AddressBooks created by the user themself.

  If false, the AddressBook and its contents should only be displayed when the user explicitly requests it or to offer it for the user to subscribe to.

- ***shareWith**: Id[AddressBookRights]|null (default: null) A map of Principal id to rights for principals this AddressBook is shared with. The principal to which this AddressBook belongs MUST NOT be in this set. This is null if the AddressBook is not shared with anyone. May be modified only if the user has the mayAdmin right. The account id for the principals may be found in the urn:ietf:params:jmap:principals:owner capability of the Account to which the AddressBook belongs.

- ***myRights**: AddressBookRights (server-set) The set of access rights the user has in relation to this AddressBook.

An **AddressBookRights** object has the following properties:

- ***mayRead**: Boolean The user may fetch the Cards and CardGroups in this AddressBook.
- ***mayWrite**: Boolean The user may create, modify or destroy all Cards and CardGroups in this AddressBook, or move them to or from this AddressBook.
- ***mayAdmin**: Boolean The user may modify sharing for this AddressBook.
- ***mayDelete**: Boolean (server-set) The user may delete the AddressBook itself. This property MUST be false if the account to which this AddressBook belongs has the *isReadOnly* property set to true.

2.1. AddressBook/get

This is a standard "/get" method as described in [RFC8620], Section 5.1. The *ids* argument may be null to fetch all at once.

2.2. AddressBook/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

2.3. AddressBook/set

This is a standard "/set" method as described in [RFC8620], Section 5.3 but with the following additional request argument:

* **onDestroyRemoveContents**: Boolean (default: false) If false, any attempt to destroy an AddressBook that still has a Card or CardGroup in it will be rejected with a addressBookHasContents SetError. If true, any Cards or CardGroups that were in the AddressBook will be destroyed.

The "shareWith" property may only be set by users that have the mayAdmin right. When modifying the shareWith property, the user cannot give a right to a principal if the principal did not already have that right and the user making the change also does not have that right. Any attempt to do so must be rejected with a forbidden SetError.

Users can subscribe or unsubscribe to an AddressBook by setting the "isSubscribed" property. The server MAY forbid users from subscribing to certain AddressBooks even though they have permission to see them, rejecting the update with a forbidden SetError.

The following extra SetError types are defined:

For "destroy":

* **addressBookHasContents**: The AddressBook has at least one Card or CardGroup assigned to it, and the "onDestroyRemoveContents" argument was false.

3. Cards

A **Card** object contains information about a person, company, or other entity. It is a JSContact Card object, as defined in RFCXXXX, with the following additional properties:

* **id**: Id (immutable; server-set) The id of the Card. The id uniquely identifies a JSCard with a particular "uid" within a particular account.

*   **addressBookId**: Id The id of the AddressBook this card belongs to. A card MUST belong to exactly one AddressBook at all times (until it is destroyed).

TODO:photos as blobs.

3.1. Card/get

This is a standard "/get" method as described in [RFC8620], Section 5.1. The *ids* argument may be null to fetch all at once.

3.2. Card/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

3.3. Card/query

This is a standard "/query" method as described in [RFC8620], Section 5.5.

3.3.1. Filtering

A **FilterCondition** object has the following properties:

*   **inAddressBook**: Id An AddressBook id. A card must be in this address book to match the condition.
*   **text**: String|null A card matches this condition if the text matches with text in the card.

If zero properties are specified on the FilterCondition, the condition MUST always evaluate to true. If multiple properties are specified, ALL must apply for the condition to be true (it is equivalent to splitting the object into one-property conditions and making them all the child of an AND filter operator).

The exact semantics for matching String fields is **deliberately not defined** to allow for flexibility in indexing implementation, subject to the following:

*   Text SHOULD be matched in a case-insensitive manner.
*   Text contained in either (but matched) single or double quotes SHOULD be treated as a **phrase search**, that is a match is required for that exact sequence of words, excluding the surrounding quotation marks. Use \", \' and \\ to match a literal ", ' and \ respectively in a phrase.
*   Outside of a phrase, white-space SHOULD be treated as dividing separate tokens that may be searched for separately in the contact, but MUST all be present for the contact to match the filter.

* Tokens MAY be matched on a whole-word basis using stemming (so for example a text search for bus would match "buses" but not "business").

3.3.2. Sorting

The following properties MUST be supported for sorting:

TODO

3.4. Card/queryChanges

This is a standard "/queryChanges" method as described in [RFC8620], Section 5.6.

3.5. Card/set

This is a standard "/set" method as described in [RFC8620], Section 5.3.

To set a new photo, the file must first be uploaded using the upload mechanism as described in [RFC8620], Section 6.1. This will give the client a valid blobId/size/type to use. The server SHOULD reject attempts to set a file that is not a recognised image type as the photo for a card.

3.6. Card/copy

This is a standard "/copy" method as described in [RFC8620], Section 5.4.

4. Card Groups

A **CardGroup** object represents a group of cards. It is a JSContact CardGroup object, as defined in RFCXXXX, with the following additional properties:

* **id**: Id (immutable; server-set) The id of the CardGroup. The id uniquely identifies a JSCard with a particular "uid" within a particular account.
* **addressBookId**: Id The id of the AddressBook this CardGroup belongs to. A CardGroup MUST belong to exactly one AddressBook at all times (until it is destroyed).

Clients should consider the set to contain any Card with a matching UID, from any account they have access to with support for the urn:ietf:params:jmap:contacts capability. UIDs that cannot be found SHOULD be ignored but preserved. For example, suppose a user adds contacts from a shared address book to their private set, then temporarily lose access to this address book. The UIDs cannot be

resolved so the contacts will disappear from the group. However, if they are given permission to access the data again the UIDs will be found and the contacts will reappear.

4.1. CardGroup/get

This is a standard "/get" method as described in [RFC8620], Section 5.1. The *ids* argument may be null to fetch all at once.

4.2. CardGroup/changes

This is a standard "/changes" method as described in [RFC8620], Section 5.2.

4.3. CardGroup/set

This is a standard "/set" method as described in [RFC8620], Section 5.3.

5. Security considerations

All security considerations of JMAP ([RFC8620]) apply to this specification. Additional considerations specific to the data types and functionality introduced by this document are described in the following subsections.

TODO

6. IANA Considerations

6.1. JMAP capability registration for "contacts"

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

Capability Name: urn:ietf:params:jmap:contacts

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, section TODO

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>.

Author's Address

Neil Jenkins (editor)
Fastmail
PO Box 234, Collins St West
Melbourne VIC 8007
Australia

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