Internet Engineering Task Force (IETF)
Request for Comments: 9670
Updates: 8620
Category: Standards Track
ISSN: 2070-1721

N. Jenkins, Ed.
Fastmail

November 2024

# JSON Meta Application Protocol (JMAP) Sharing

## Abstract

This document specifies a data model for sharing data between users using the JSON Meta Application Protocol (JMAP). Future documents can reference this document when defining data types to support a consistent model of sharing.

## 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/rfc9670.



## Table of Contents

## 1. Introduction

The JSON Meta Application Protocol (JMAP) [RFC8620] is a generic protocol for synchronizing data, such as mail, calendars, or contacts, between a client and a server. It is optimized for mobile and web environments and provides a consistent interface to query, read, and modify different data types, including comprehensive error handling.

This specification defines a data model to represent entities in a collaborative environment and a framework for sharing data between them that can be used to provide a consistent sharing model for different data types. It does not define _what_ may be shared or the granularity of permissions, as this will depend on the data in question.

### 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.

Examples of API exchanges only show the methodCalls array of the Request object or the methodResponses array of the Response object. For compactness, the rest of the Request/Response object is omitted.

### 1.2. Terminology

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

The terms "Principal" and "ShareNotification" (with this specific capitalization) are used to refer to the data types defined in this

document and instances of those data types.

### 1.3. Data Model Overview

A Principal (see Section 2) represents an individual, team, or resource (e.g., a room or projector). The object contains information about the entity being represented, such as a name, description, and time zone. It may also hold domain-specific information. A Principal may be associated with zero or more Accounts (see [RFC8620], Section 1.6.2) containing data belonging to the Principal. Managing the set of Principals within a system is out of scope for this specification, as it is highly domain specific. It is likely to map directly from a directory service or other user management system.

Data types may allow users to share data with others by assigning permissions to Principals. When a user's permissions are changed, a ShareNotification object is created for them so a client can inform the user of the changes.

### 1.4. Subscribing to Shared Data

Permissions determine whether a user _may_ access data but not whether they _want_ to. Some shared data is of equal importance as the user's own, while other data is just there should the user wish to explicitly go find it. Clients will often want to differentiate the two. For example, a company may share mailing list archives for all departments with all employees, but a user may only generally be interested in the few they belong to. They would have _permission_ to access many mailboxes but can _subscribe_ to just the ones they care about. The client would provide separate interfaces for reading mail in subscribed mailboxes and browsing all mailboxes they have permission to access in order to manage those that they are subscribed to.

The JMAP Session object (see [RFC8620], Section 2) is defined to include an object in the "accounts" property for every Account that the user has access to. Collaborative systems may share data between a very large number of Principals, most of which the user does not care about day to day. For servers implementing this specification, the Session object MUST only include Accounts where either the user is subscribed to at least one record (see [RFC8620], Section 1.6.3) in the Account or the Account belongs to the user. StateChange events ([RFC8620], Section 7.1) for changes to data SHOULD only be sent for data the user has subscribed to and MUST NOT be sent for any Account where the user is not subscribed to any records in the Account, except where that Account belongs to the user.

The server MAY reject the user's attempt to subscribe to some resources even if they have permission to access them (e.g., a calendar representing a location).

A user can query the set of Principals they have access to with "Principal/query" (see Section 2.4). The Principal object will contain an Account object for all Accounts where the user has permission to access data for that Principal, even if they are not yet subscribed.

### 1.5. 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.

#### 1.5.1. urn:ietf:params:jmap:principals

The urn:ietf:params:jmap:principals capability represents support for the Principal and ShareNotification 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's "accountCapabilities" property is an object that MUST contain the following information on server capabilities and permissions for that Account:

*currentUserPrincipalId*: Id|null
The id of the Principal in this Account that corresponds to the user fetching this object, if any.

### 1.5.2. urn:ietf:params:jmap:principals:owner

The URI urn:ietf:params:jmap:principals:owner is solely used as a key in an Account's "accountCapabilities" property. It does not appear in the JMAP Session capabilities -- support is indicated by the urn:ietf:params:jmap:principals URI being present in the session capabilities.

If urn:ietf:params:jmap:principals:owner is a key in an Account's "accountCapabilities" property, that Account (and the data therein) is owned by a Principal. Some Accounts may not be owned by a Principal (e.g., the Account that contains the data for the Principals themselves), in which case this property is omitted.

The value of this property is an object with the following properties:

*accountIdForPrincipal*: Id
The id of an Account with the urn:ietf:params:jmap:principals capability that contains the corresponding Principal object.

*principalId*:Id
The id of the Principal that owns this Account.

## 2. Principals

A Principal represents an individual, a group, a location (e.g., a room), a resource (e.g., a projector), or another entity in a collaborative environment. Sharing in JMAP is generally configured by assigning rights to certain data within an Account to other Principals. For example, a user may assign permission to read their calendar to a Principal representing another user or their team.

In a shared environment, such as a workplace, a user may have access to a large number of Principals.

In most systems, the user will have access to a single Account containing Principal objects. In some situations, for example, when aggregating data from different places, there may be multiple Accounts containing Principal objects.

A *Principal* object has the following properties:

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

*type*: String
This MUST be one of the following values:

* "individual": This represents a single person.
* "group": This represents a group of other Principals.

* "resource": This represents some resource, e.g., a projector.
* "location": This represents a location.
* "other": This represents some other undefined Principal.

*name*: String
The name of the Principal, e.g., "Jane Doe" or "Room 4B".

*description*: String|null
A longer description of the Principal, for example, details about the facilities of a resource, or null if no description is available.

*email*: String|null
An email address for the Principal, or null if no email is available. If given, the value MUST conform to the "addr-spec" syntax, as defined in [RFC5322], Section 3.4.1.

*timeZone*: String|null
The time zone for this Principal, if known. If not null, the value MUST be a time zone name from the IANA Time Zone Database [IANA-TZDB].

*capabilities*: String[Object] (server-set)
A map of JMAP capability URIs to domain-specific information about the Principal in relation to that capability, as defined in the document that registered the capability.

*accounts*: Id[Account]|null (server-set)
A map of Account id to Account object for each JMAP Account containing data for this Principal that the user has access to, or null if none.

## 2.1. Principal/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.

## 2.2. Principal/changes

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

| Note: Implementations backed by an external directory may be
| unable to calculate changes. In this case, they will always
| return a "cannotCalculateChanges" error as described in the
| core JMAP specification.

## 2.3. Principal/set

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

Managing Principals is likely tied to a directory service or some other vendor-specific solution. This management may occur out of band or via an additional capability defined elsewhere. Allowing direct user modification of properties has security considerations, as noted in Section 6. A server MUST reject any change it doesn't allow with a "forbidden" SetError.

Where a server does support changes via this API, it SHOULD allow an update to the "name", "description", and "timeZone" properties of the Principal with the same id as the "currentUserPrincipalId" in the Account capabilities. This allows the user to update their own details.

## 2.4. Principal/query

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

#### 2.4.1. Filtering

A *FilterCondition* object has the following properties, all of which are optional:

*accountIds*: String[]
: A list of Account ids. The Principal matches if any of the ids in this list are keys in the Principal's "accounts" property (i.e., if any of the Account ids belong to the Principal).

*email*: String
: The email property of the Principal contains the given string.

*name*: String
: The name property of the Principal contains the given string.

*text*: String
: The name, email, or description property of the Principal contains the given string.

*type*: String
: The type must be exactly as given to match the condition.

*timeZone*: String
: The timeZone must be exactly as given to match the condition.

All given conditions in the FilterCondition object must match for the Principal to match.

Text matches for "contains" SHOULD be simple substring matches.

### 2.5. Principal/queryChanges

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

> Note: Implementations backed by an external directory may be unable to calculate changes. In this case, they will always return a "cannotCalculateChanges" error as described in the core JMAP specification.

## 3. ShareNotifications

The ShareNotification data type records when the user's permissions to access a shared object changes. ShareNotifications are only created by the server; users cannot create them explicitly. They are stored in the same Account as the Principals.

Clients may present the list of notifications to the user and allow the user to dismiss them. To dismiss a notification, use a standard "/set" call to destroy it.

The server SHOULD create a ShareNotification whenever the user's permissions change on an object. It MAY choose not to create a notification for permission changes to a group Principal, even if the user is in the group, if this is more likely to be overwhelming than helpful, or if it would create excessive notifications within the system.

The server MAY limit the maximum number of notifications it will store for a user. When the limit is reached, any new notification will cause the previously oldest notification to be automatically

deleted.

The server MAY coalesce notifications if appropriate or remove notifications after a certain period of time or that it deems are no longer relevant.

A *ShareNotification* object has the following properties:

*id*: String (immutable; server-set)
The id of the ShareNotification.

*created*: UTCDate (immutable; server-set)
The time this notification was created.

*changedBy*: Entity (immutable; server-set)
Who made the change. An *Entity* object has the following properties:

*name*: String
The name of the entity who made the change.
*email*: String|null
The email of the entity who made the change, or null if no email is available.
*principalId*: Id|null
The id of the Principal corresponding to the entity who made the change, or null if no associated Principal.

*objectType*: String (immutable; server-set)
The name of the data type for the object whose permissions have changed, as registered in the IANA "JMAP Data Types" registry [IANA-JMAP], e.g., "Calendar" or "Mailbox".

*objectAccountId*: Id (immutable; server-set)
The id of the Account where this object exists.

*objectId*: Id (immutable; server-set)
The id of the object that this notification is about.

*oldRights*: String[Boolean]|null (immutable; server-set)
The "myRights" property of the object for the user before the change.

*newRights*: String[Boolean]|null (immutable; server-set)
The "myRights" property of the object for the user after the change.

*name*: String (immutable; server-set)
The name of the object at the time the notification was made. Determining the name will depend on the data type in question. For example, it might be the "title" property of a CalendarEvent or the "name" of a Mailbox. The name is to show users who have had their access rights to the object removed what it is that they can no longer access.

## 3.1. ShareNotification/get

This is a standard "/get" method as described in [RFC8620], Section 5.1.

## 3.2. ShareNotification/changes

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

## 3.3. ShareNotification/set

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

Only destroy is supported; any attempt to create/update MUST be rejected with a "forbidden" SetError.

### 3.4. ShareNotification/query

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

#### 3.4.1. Filtering

A *FilterCondition* object has the following properties, all of which are optional:

*after*: UTCDate|null
: The creation date must be on or after this date to match the condition.

*before*: UTCDate|null
: The creation date must be before this date to match the condition.

*objectType*: String
: The objectType value must be identical to the given value to match the condition.

*objectAccountId*: Id
: The objectAccountId value must be identical to the given value to match the condition.

All given conditions in the FilterCondition object must match for the ShareNotification to match.

#### 3.4.2. Sorting

The "created" property MUST be supported for sorting.

### 3.5. ShareNotification/queryChanges

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

## 4. Framework for Shared Data

Shareable data types MUST define the following three properties:

*isSubscribed*: Boolean
: The value true indicates that the user wishes to subscribe to see this data. The value false indicates that the user does not wish to subscribe to see this data. The initial value for this property when data is shared by another user is implementation dependent, although data types may give advice on appropriate defaults.

*myRights*: String[Boolean]
: The set of permissions the user currently has. Appropriate permissions are domain specific and must be defined per data type. Each key is the name of a permission defined for that data type. The value for the key is true if the user has the permission or false if they do not.

*shareWith*: Id[String[Boolean]]|null
: The value of this property is null if the data is not shared with anyone. Otherwise, it is a map where each key is the id of a Principal with which this data is shared, and the value associated

with that key is the rights to give that Principal, in the same format as the "myRights" property. The Account id for the Principal id can be found in the capabilities of the Account this object is in (see Section 1.5.2).

Users with appropriate permission may set this property to modify who the data is shared with. The Principal that owns the Account that this data is in MUST NOT be in the map, since the owner's rights are implicit.

## 4.1. Example

Suppose we are designing a data model for a very simple to-do list. There is a Todo data type representing a single item to do, each of which belongs to a single TodoList. The specification makes the TodoLists shareable by referencing this document and defining the common properties.

First, it would define a set of domain-specific rights. For example, a TodoListRights object may have the following properties:

*mayRead*: Boolean
The user may fetch this TodoList and any Todos that belong to this TodoList.

*mayWrite*: Boolean
The user may create, update, or destroy Todos that belong to this TodoList and may change the "name" property of this TodoList.

*mayAdmin*: Boolean
The user may see and modify the "myRights" property of this TodoList and may destroy this TodoList.

Then in the TodoList data type, we would include the three common properties described in Section 4, in addition to any type-specific properties (like "name" in this case):

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

*name*: String
A name for this list of Todos.

*isSubscribed*: Boolean
True if the user has indicated they wish to see this list. If false, clients should not display this TodoList with the user's other TodoLists but should provide a means for users to see and subscribe to all TodoLists that have been shared with them.

*myRights*: TodoListRights
The set of permissions the user currently has for this TodoList.

*shareWith*: Id[TodoListRights]|null
If not shared with anyone, the value is null. Otherwise, it's a map where the keys are Principal ids and the values are the rights given to those Principals. Users with the "mayAdmin" right may set this property to modify who the data is shared with. The Principal that owns the Account that this data is in MUST NOT be in the map; their rights are implicit.

We would also define a new Principal capability with two properties:

*accountId*: Id|null
The accountId containing the Todo data for this Principal, if it has been shared with the requesting user.

*mayShareWith*: Boolean
   The user may give this Principal permission to access a TodoList.

A client wishing to let the user configure sharing would look at the "capabilities" for the Account containing the user's Todo data and find the "urn:ietf:params:jmap:principals:owner" property, as per Section 1.5.2.  For example, the JMAP Session object might contain:

```
{
  "accounts": {
    "u12345678": {
      "name": "jane.doe@example.com",
      "isPersonal": true,
      "isReadOnly": false,
      "accountCapabilities": {
        "urn:com.example:jmap:todo": {},
        "urn:ietf:params:jmap:principals:owner": {
          "accountIdForPrincipal": "u33084183",
          "principalId": "P105aga511jaa"
        }
      }
    },
    ...
  },
  ...
}
```

Figure 1: Part of a JMAP Session Object

From this, the client now knows which Account has the Principal data, and it can fetch the list of Principals and offer to share it with the user by making an API request like this:

```
[[ "Principal/get", {
  "accountId": "u33084183",
  "ids": null
}, "0" ]]
```

Figure 2: "methodCalls" Property of a JMAP Request

Here's an example response (where "Joe Bloggs" is another user that this user could share their TodoList with; Joe has not shared any of their own data with this user, so the "accounts" property is null):

```
[[ "Principal/get", {
  "accountId": "u33084183",
  "state": "7b8eff5zz",
  "list": [{
    "id": "P2342fnddd20",
    "type": "individual",
    "name": "Joe Bloggs",
    "description": null,
    "email": "joe.bloggs@example.com",
    "timeZone": "Australia/Melbourne",
    "capabilities": {
        "urn:com.example:jmap:todo": {
            "accountId": null,
            "mayShareWith": true
        }
    },
    "accounts": null
  }, {
    "id": "P674pp24095qo49pr",
    "name": "Board room",
    "type": "location",
    ...
```

```
  }, ... ],
  "notFound": []
}, "0" ]]
```

Figure 3: "methodResponses" Property of a JMAP Response

A TodoList can be shared with "Joe Bloggs" by updating its shareWith property, as in this example request:

```
[[ "TodoList/set", {
  "accountId": "u12345678",
  "update": {
    "tl01n231": {
        "shareWith": {
            "P2342fnddd20": {
                "mayRead": true,
                "mayWrite": true,
                "mayAdmin": false
            }
        }
    }
  }
}, "0" ]]
```

Figure 4: "methodCalls" Property of a JMAP Request

## 5. Internationalization Considerations

Experience has shown that unrestricted use of Unicode can lead to problems such as inconsistent rendering, users reading text and interpreting it differently than intended, and unexpected results when copying text from one location to another. Servers MAY choose to mitigate this by restricting the set of characters allowed in otherwise unconstrained String fields. The FreeformClass, as documented in [RFC8264], Section 4.3, might be a good starting point for this.

Attempts to set a value containing code points outside of the permissible set can be handled in a few ways by the server. The first option is to simply strip the forbidden characters and store the resulting string. This is likely to be appropriate for control characters, for example, where they can end up in data accidentally due to copy-and-paste issues and are probably invisible to the end user. JMAP allows the server to transform data on create/update, as long as any changed properties are returned to the client in the "/set" response so it knows what has changed, as per [RFC8620], Section 5.3. Alternatively, the server MAY just reject the create/update with an "invalidProperties" SetError.

## 6. Security Considerations

All security considerations of JMAP [RFC8620] apply to this specification. Additional considerations are detailed below.

### 6.1. Spoofing

Allowing users to edit their own Principal’s name (and, to a lesser extent, email, description, or type) could allow a user to change their Principal to look like another user in the system, potentially tricking others into sharing private data with them. Servers may choose to forbid this and SHOULD keep logs of such changes to provide an audit trail.

Note that simply forbidding the use of a name already in the system is insufficient protection, as a malicious user could still change their name to something easily confused with the existing name by

using trivial misspellings or visually similar Unicode characters.

### 6.2. Unnoticed Sharing

Sharing data with another user allows someone to turn a transitory account compromise (e.g., brief access to an unlocked or logged-in client) into a persistent compromise (by setting up sharing with a user that is controlled by the attacker). This can be mitigated by requiring further authorization for configuring sharing or sending notifications to the sharer via another channel whenever a new permission is added.

### 6.3. Denial of Service

By creating many changes to the sharing status of objects, a user can cause many ShareNotifications to be generated, which could lead to resource exhaustion. Servers can mitigate this by coalescing multiple changes to the same object into a single notification, limiting the maximum number of notifications it stores per user and/or rate-limiting the changes to sharing permissions in the first place. Automatically deleting older notifications after reaching a limit can mean the user is not made aware of a sharing change, which can itself be a security issue. For this reason, it is better to coalesce changes and use other mitigation strategies.

### 6.4. Unauthorized Principals

The set of Principals within a shared environment MUST be strictly controlled. If adding a new Principal is open to the public, risks include:

* An increased risk of a user accidentally sharing data with an unintended person.
* An attacker sharing unwanted or offensive information with the user.
* An attacker sharing items with spam content in the names in order to generate ShareNotification objects, which are likely to be prominently displayed to the user receiving them.

## 7. IANA Considerations

### 7.1. JMAP Capability Registration for "principals"

IANA has registered "principals" in the "JMAP Capabilities" registry as follows:

Capability Name: urn:ietf:params:jmap:principals
Intended Use: common
Change Controller: IETF
Security and Privacy Considerations: RFC 9670, Section 6
Reference: RFC 9670

### 7.2. JMAP Capability Registration for "principals:owner"

IANA has registered "principals:owner" in the "JMAP Capabilities" registry as follows:

Capability Name: urn:ietf:params:jmap:principals:owner
Intended Use: common
Change Controller: IETF
Security and Privacy Considerations: RFC 9670, Section 6
Reference: RFC 9670

### 7.3. JMAP Data Type Registration for "Principal"

IANA has registered "Principal" in the "JMAP Data Types" registry as

follows:

Type Name: Principal
Can Reference Blobs: No
Can Use for State Change: Yes
Capability: urn:ietf:params:jmap:principals
Reference: RFC 9670

### 7.4. JMAP Data Type Registration for "ShareNotification"

IANA has registered "ShareNotification" in the "JMAP Data Types" registry as follows:

Type Name: ShareNotification
Can Reference Blobs: No
Can Use for State Change: Yes
Capability: urn:ietf:params:jmap:principals
Reference: RFC 9670

## 8. References

### 8.1. 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>.

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

[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>.


### 8.2. Informative References


[IANA-JMAP] IANA, "JMAP Data Types", <https://www.iana.org/assignments/jmap>.

[IANA-TZDB] IANA, "Time Zone Database", <https://www.iana.org/time-zones>.

[RFC8264] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/info/rfc8264>.


## 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