JMAP
Internet-Draft
Intended status: Standards Track
Expires: 22 June 2026

M. De Gennaro
Stalwart Labs

19 December 2025

JMAP Object Metadata
draft-ietf-jmap-metadata-01

Abstract

This document defines a new extension to the JSON Meta Application Protocol (JMAP) that introduces a standardized mechanism for managing metadata associated with JMAP objects. The JMAP Object Metadata extension allows clients and servers to store, retrieve, and synchronize arbitrary metadata and annotations on any JMAP data type in a consistent manner. It defines a generic annotation model as well as specific mappings for accessing IMAP metadata and WebDAV “dead properties” where applicable. This extension facilitates interoperability between JMAP and existing metadata frameworks while allowing vendor-specific extensions in a structured and discoverable way.

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 June 2026.

Copyright Notice

Table of Contents

1. Introduction

JMAP ([RFC8620] — JSON Meta Application Protocol) 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 aims to provide a consistent interface to different data types.

Metadata and annotations are auxiliary data elements that provide additional context, user-defined properties, or system-specific information about primary data objects. These mechanisms have proven valuable in existing protocols: IMAP [RFC3501] provides the METADATA extension [RFC5464] for associating arbitrary key-value pairs with mailboxes and the server, while WebDAV [RFC4918] supports "dead properties" that allow clients to store custom XML properties on resources. Such metadata capabilities enable a wide range of use cases, including user annotations, application-specific settings, collaborative tagging, and protocol bridging.

This document defines a standardized approach to managing metadata within JMAP. The specification introduces a generic annotation mechanism that can be applied to any JMAP object type, allowing both common metadata properties and vendor-specific extensions. This provides a consistent interface for clients to store and retrieve supplementary information associated with emails, contacts, calendar events, and other JMAP data types.

For servers that implement multiple protocols, this specification also defines optional specialized metadata types: ImapMetadata for accessing IMAP METADATA extension entries through JMAP, and WebDavMetadata for exposing WebDAV dead properties via JMAP interfaces. These specialized types facilitate protocol interoperability and allow clients to manage metadata across different protocol boundaries.

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.

1.2. Addition to the Capabilities Object

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

1.2.1. urn:ietf:params:jmap:metadata

This capability represents support for the Metadata data type and associated API methods. Servers that include this capability provide the ability to create, retrieve, update, and query Metadata objects.

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:

dataTypes: String[]|null A list of JMAP data types for which the server supports metadata operations. A value of null indicates support for all data types. When specified as an array, only the listed data types can have Metadata objects associated with them.

metadataTypes: String[] A list of metadata type identifiers (values of the @type property) for which the server supports metadata operations. Only the listed metadata types can be created or retrieved.

maxDepth: UnsignedInt|null Maximum depth of nested vendor-specific metadata properties that can be set or retrieved. A depth of 1 indicates only flat properties are supported. A depth of 2 allows one level of nesting, and so forth. A value of null indicates no server-enforced limit on nesting depth. This limitation applies to the structure of vendor-defined nested metadata properties within Annotation objects and does not affect the standard properties defined in this specification.

maySetPrivate: Boolean (default: true) Indicates whether the authenticated user has permission to create private metadata objects (isPrivate: true) in this account. If false, the user can only create shared metadata objects. Attempts to create private metadata when this capability is false MUST be rejected with a "forbidden" SetError.

2. Metadata Object

The Metadata object has a collection of properties, as specified in the following sections. Properties are specified as being either mandatory or optional. Optional properties may have a default value if explicitly specified in the property definition.

2.1. Object Types

This specification defines three metadata object types: Annotation, ImapMetadata, and WebDavMetadata. The Annotation type provides a generic mechanism for attaching metadata to any JMAP object and supports arbitrary vendor-specific properties. The ImapMetadata and WebDavMetadata types are specialized metadata objects that provide structured access to IMAP METADATA extension [RFC5464] entries and WebDAV [RFC4918] dead properties, respectively, enabling protocol interoperability for servers that support multiple protocols.

Additional specifications MAY define further metadata types as needed to address specific use cases or protocol requirements. Such specifications MUST define a unique type identifier and specify which JMAP data types the metadata type may be associated with.

Metadata objects MUST name their type in the @type property. If not specified, the type is assumed to be "Annotation".

2.1.1. Annotation

An Annotation represents general-purpose metadata that can be attached to any JMAP object. Annotations provide a flexible mechanism for storing supplementary information about JMAP data objects without modifying the objects themselves. This metadata type is suitable for user comments, application-specific tags, workflow states, collaborative annotations, and other extensible metadata requirements.

The Annotation type is designed to be extensible through vendor-specific properties. This allows vendors and third-party applications to add custom metadata fields while maintaining interoperability with the core JMAP metadata system. Vendors can define domain-namespaced properties to avoid naming conflicts and ensure that their extensions do not interfere with standard properties or extensions from other vendors.

Common use cases for Annotations include: storing user notes or comments on emails or calendar events; maintaining application-specific flags or states; recording collaborative review or approval status; tracking custom workflow stages; and preserving client-specific UI preferences or cached computations.

The Annotation type MUST allow vendor-specific properties to be added beyond the common properties defined in this specification, subject to the naming and structure requirements specified in Vendor-Specific Properties (Section 2.2.1.6).

The @type (Section 2.2.1.1) property value MUST be "Annotation".

2.1.2. IMAP Metadata

ImapMetadata represents metadata associated with IMAP mailboxes as defined by the IMAP METADATA extension [RFC5464]. This metadata type provides a JMAP interface to IMAP metadata entries, allowing clients to read and potentially modify IMAP metadata through the JMAP protocol. This facilitates migration scenarios, protocol bridging, and unified metadata management for servers that support both IMAP and JMAP.

The IMAP METADATA extension defines a hierarchical namespace of metadata entries associated with mailboxes, organized under "/shared" and "/private" prefixes. Shared metadata entries are visible to all users with appropriate access to the mailbox, while private metadata entries are visible only to the authenticated user. The ImapMetadata type maps these IMAP concepts onto the JMAP metadata model.

Servers that support ImapMetadata MUST provide read access to IMAP metadata through the Metadata/get method. Write access (the ability to create, update, or destroy ImapMetadata objects through Metadata/set) is OPTIONAL. Servers that provide read-only access to IMAP metadata MUST reject modification attempts with an appropriate SetError (typically "forbidden"). Servers SHOULD document whether they provide read-only or read-write access to ImapMetadata.

This metadata type can only be associated with JMAP Mailbox objects as defined in Section 2 of [RFC8621]. Attempts to create ImapMetadata objects with a relatedType other than "Mailbox" MUST be rejected with an "invalidProperties" SetError.

The @type (Section 2.2.1.1) property value MUST be "ImapMetadata".

2.1.3. WebDAV Metadata

WebDavMetadata represents metadata associated with WebDAV resources in the form of "dead properties" as defined in Section 4 of [RFC4918]. Dead properties are arbitrary XML properties that clients can set on WebDAV resources to store metadata. This metadata type enables JMAP clients to access and potentially manipulate WebDAV dead properties, facilitating protocol interoperability and unified metadata management on servers that support both WebDAV and JMAP.

WebDAV dead properties are identified by their expanded XML name, which consists of a namespace URI and a local name. The WebDavMetadata type preserves this structure by using the expanded-name format for property keys: "{namespace-uri}localname". This ensures that WebDAV properties can be accurately represented and manipulated through JMAP without loss of namespace information.

Servers that support WebDavMetadata MUST provide read access to WebDAV dead properties through the Metadata/get method. Write access (the ability to create, update, or destroy WebDavMetadata objects through Metadata/set) is OPTIONAL. Servers that provide read-only access to WebDAV dead properties MUST reject modification attempts with an appropriate SetError (typically "forbidden"). Servers SHOULD document whether they provide read-only or read-write access to WebDavMetadata.

This metadata type can be associated with Calendar [I-D.ietf-jmap-calendars], CalendarEvent [I-D.ietf-jmap-calendars], AddressBook [RFC9610], ContactCard [RFC9610], and FileNode [I-D.ietf-jmap-filenode] data types, corresponding to the WebDAV-based protocols such as CalDAV [RFC4791], CardDAV [RFC6352], and WebDAV file storage. Servers MUST enforce that WebDavMetadata objects are only created for parent types that correspond to WebDAV

resources. Attempts to create WebDavMetadata objects with inappropriate relatedType values MAY be rejected with an "invalidProperties" SetError.

The @type (Section 2.2.1.1) property value MUST be "WebDavMetadata".

2.2. Properties

2.2.1. Common Properties

2.2.1.1. @type

Type: String (mandatory)

This specifies the type that this object represents. The allowed value differs by object type and is defined in Sections Section 2.1.1, Section 2.1.2, and Section 2.1.3.

2.2.1.2. id

Type: Id (server-set)

The id of the Metadata object. This identifier is unique within the scope of the account and is assigned by the server upon creation.

2.2.1.3. relatedType

Type: String (mandatory)

The JMAP data type of the object to which this Metadata object is associated, e.g., "Email", "ContactCard", "Mailbox", "CalendarEvent". The relatedType determines the kind of object identified by the relatedId property.

2.2.1.4. relatedId

Type: Id (mandatory)

The id of the JMAP object to which this Metadata object is associated. This id MUST correspond to a valid object of the type specified in relatedType within the same account.

2.2.1.5. isPrivate

Type: Boolean (default: false)

Indicates whether the Metadata object is private to the authenticated user or shared among all users with access to the related object. If true, the Metadata object represents user-specific metadata that is visible only to the user who created it. If false or omitted, the Metadata object is considered shared and can be accessed by other users with appropriate permissions on the related object, as defined in Access Control (Section 5).

The isPrivate property enables per-user metadata on shared objects. Multiple users may each have their own private Metadata objects associated with the same related object and metadata type. When a client queries for Metadata objects with isPrivate: true, the server MUST return only those private Metadata objects that were created by the currently authenticated user. Private Metadata objects created by other users MUST NOT be visible to the authenticated user, even if they have access to the related object.

Internally, the server may store multiple private Metadata objects for the same combination of relatedType, relatedId, and @type (one for each user who has set private metadata). However, from the perspective of any given authenticated user, there is at most one visible private Metadata object for each related-type combination— specifically, the private Metadata object that they themselves created. Other users' private Metadata objects for the same related object and type are not visible to them.

For shared Metadata objects (isPrivate: false), there is at most one Metadata object per combination of relatedType, relatedId, and @type, and it is visible to all users with appropriate permissions on the related object.

For ImapMetadata objects, this property determines whether the metadata corresponds to IMAP entries under the "/private/" or "/shared/" prefix hierarchy.

2.2.1.6. Vendor-Specific Properties

Vendors MAY add additional properties to Metadata objects to support their custom features or requirements. To prevent naming conflicts, the names of vendor-specific properties MUST be prefixed by a domain name controlled by the vendor followed by a colon, e.g., example.com:customField or vendor.org:internalState. The use of domain name prefixes ensures global uniqueness and allows multiple vendors to extend the metadata schema without interference.

If the value of a vendor-specific property is itself a Metadata object (nested object), it either MUST include an @type property to identify its type, or the property definition MUST explicitly specify

that a type designator is not required for that particular property. If a type is specified for a nested vendor-specific object, the type name MUST also be prefixed with a domain name controlled by the vendor, e.g., example.com:CustomMetadataType.

Vendors are strongly encouraged to register any new property values or extensions that are useful to other systems as well, rather than use a vendor-specific prefix. Registration with IANA or documentation in public specifications enhances interoperability and allows broader adoption of useful metadata extensions.

Server implementations MUST preserve vendor-specific properties that they do not recognize when updating Metadata objects, unless the update explicitly removes those properties. This ensures that metadata from one vendor's client is not inadvertently lost when another client (or the server itself) modifies other properties of the Metadata object.

Annotation type Metadata objects MUST allow vendor-specific properties to be added beyond the common properties defined above. This extensibility mechanism enables vendors and third-party applications to store custom metadata without requiring standardization of every use case.

2.2.2. ImapMetadata Properties

In addition to the common Metadata object properties (Section 2.2.1), an ImapMetadata object has the following properties:

2.2.2.1. metadata

Type: [String]String (default: {})

The IMAP METADATA extension [RFC5464] defines a mechanism for clients to store and retrieve arbitrary key-value metadata associated with IMAP mailboxes. Metadata entries are organized in a hierarchical namespace with two top-level prefixes: "/private/" for per-user private metadata and "/shared/" for metadata shared among all users with access to the mailbox.

This property contains the key-value pairs of IMAP metadata entries associated with the mailbox. The keys in this map represent IMAP metadata entry names with the "/private/" or "/shared/" prefix removed. The values are the corresponding metadata values as strings. The interpretation of keys depends on the value of the isPrivate property:

* If isPrivate is true, the keys represent metadata entries under the "/private/" prefix. For example, a key "comment" with value "Important mailbox" corresponds to the IMAP metadata entry "/private/comment" with that value. A key "vendor/example.com/setting" would correspond to "/private/vendor/example.com/setting".

* If isPrivate is false, the keys represent metadata entries under the "/shared/" prefix. A key "comment" with value "Team mailbox" would correspond to "/shared/comment". For example, if an ImapMetadata object has isPrivate set to true and the metadata property contains {"comment": "My notes", "vendor/acme.example/color": "blue"}, this represents IMAP metadata entries "/private/comment" with value "My notes" and "/private/vendor/acme.example/color" with value "blue".

Servers MUST ensure that the IMAP metadata namespace conventions are preserved. When retrieving IMAP metadata, servers SHOULD strip the "/private/" or "/shared/" prefix based on the isPrivate property value. When setting IMAP metadata through this interface, servers MUST prepend the appropriate prefix before storing the entry in the IMAP metadata store.

Empty string values are permitted and represent IMAP metadata entries that exist but have no value. To delete an IMAP metadata entry through this interface, clients should remove the corresponding key from the metadata map entirely.

2.2.3. WebDavMetadata Properties

In addition to the common Metadata object properties (Section 2.2.1), an WebDavMetadata object has the following properties:

2.2.3.1. metadata

Type: [String]String (default: {})

WebDAV [RFC4918] defines "dead properties" as arbitrary properties set by clients on WebDAV resources. Unlike "live properties" which have their semantics enforced by the server, dead properties are stored and returned verbatim without server interpretation. Dead properties are identified by XML expanded names, which consist of a namespace URI and a local name.

This property contains the key-value pairs of WebDAV dead properties associated with the resource. The keys MUST be in the expanded-name format: "{namespace-uri}localname". For example, a property with namespace URI "http://example.com/ns" and local name "priority" would be represented as the key "{http://example.com/ns}priority".

The values are the string representations of the dead properties. For properties that contain simple text content, the value is that text content. For properties that contain complex XML structure or content that cannot be directly represented as a simple string, the value MUST be the serialized XML representation of the property's content. The serialization SHOULD preserve the XML structure including elements, attributes, and namespaces, but typically excludes the outer property element itself (since the property name is already represented in the key).

For example, a WebDavMetadata object might have a metadata property containing:

```
{
  "{http://example.com/ns}priority": "high",
  "{http://example.com/ns}reviewedBy": "alice@example.com",
  "{DAV:}displayname": "Project Documents",
  "{http://example.com/ns}complexdata":
     "<item><name>Test</name><value>123</value></item>"
}
```

This represents four WebDAV dead properties: three with simple text content and one with XML structure serialized as a string.

Servers MUST preserve the namespace and local name structure of WebDAV properties. When synchronizing between JMAP and WebDAV interfaces, servers MUST maintain consistency such that setting a property through one interface makes it visible through the other interface (subject to server support for write operations).

The XML serialization format allows round-tripping of complex WebDAV properties through the JMAP interface. Clients that need to work with structured WebDAV properties can parse the XML string values to access the internal structure.

3. Metadata Methods

3.1. Metadata/set

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

The Metadata/set method enforces a uniqueness constraint to ensure consistent metadata management. From the perspective of an authenticated user, their account MUST NOT contain multiple visible Metadata objects with the same combination of relatedType, relatedId, @type, and isPrivate values. This constraint ensures that for any given related object and metadata type, the user sees at most one private Metadata object (their own) and at most one shared Metadata object.

For shared metadata (isPrivate: false), the server MUST enforce global uniqueness: only one shared Metadata object may exist for each combination of relatedType, relatedId, and @type across all users.

For private metadata (isPrivate: true), the server MUST enforce per-user uniqueness: each user may have at most one private Metadata object for each combination of relatedType, relatedId, and @type. The server internally may store multiple private Metadata objects for the same related-type combination (one per user), but each user can only see and modify their own.

If a client attempts to create a Metadata object that would violate this uniqueness constraint for the authenticated user, the server MUST reject the create operation with an "alreadyExists" SetError. The SetError SHOULD include a description indicating that a Metadata object with the specified combination of properties already exists, and MAY include an "existingId" property containing the id of the existing Metadata object. Clients that wish to modify existing metadata should use the update operation rather than attempting to create a duplicate.

Similarly, if an update operation would change the relatedType, relatedId, @type, or isPrivate properties such that the resulting Metadata object would conflict with an existing object visible to the authenticated user, the server MUST reject the update with an "alreadyExists" SetError.

If the maySetPrivate capability is false for the account and a client attempts to create or update a Metadata object with isPrivate: true, the server MUST reject the operation with a "forbidden" SetError.

Servers SHOULD enforce quota limits when processing Metadata/set requests. If a create or update operation would cause the account’s metadata storage to exceed its quota, the server MUST reject the operation with an "overQuota" SetError. The SetError MAY include additional information about the quota limit and current usage in its description property. Quota enforcement is discussed further in Section 6.

Servers MUST validate that the relatedType and relatedId values reference a valid existing object of the appropriate type. If the related object does not exist or the relatedType is not recognized, the server SHOULD reject the operation with an "invalidProperties" SetError, with the "properties" field including "relatedType" and/or "relatedId" as appropriate.

For ImapMetadata objects, servers MUST verify that the relatedType is "Mailbox" (Section 2 of [RFC8621]). For WebDavMetadata objects, servers MUST verify that the relatedType is one of the supported WebDAV-backed types (Calendar [I-D.ietf-jmap-calendars], CalendarEvent [I-D.ietf-jmap-calendars], AddressBook [RFC9610], ContactCard [RFC9610], FileNode [I-D.ietf-jmap-filenode], or other appropriate types as defined by the server).

3.2. Metadata/get

This is a standard "/get" method as described in Section 5.1 of [RFC8620]. The "ids" argument MAY be null to fetch all Metadata objects in the account at once.

3.3. Metadata/changes

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

In addition to the standard arguments defined in Section 5.2 of [RFC8620], the Metadata/changes method accepts the following optional arguments:

filterRelatedType: String|null (default: null) If specified, the response MUST include only changes to Metadata objects whose relatedType property equals the specified value. This allows clients to efficiently synchronize metadata changes for a specific JMAP data type (e.g., only changes to metadata associated with Email objects). If null, changes to Metadata objects of all related types are returned.

filterMetadataType: String[]|null (default: null) If specified, the response MUST include only changes to Metadata objects whose @type property value is in the specified array. This allows clients to efficiently synchronize changes for specific metadata types (e.g., only Annotation changes, or only ImapMetadata and WebDavMetadata changes). If null, changes to Metadata objects of all metadata types are returned.

These filtering arguments do not affect the state string returned in the response. The state string represents the complete state of all Metadata objects in the account. However, the "created", "updated", and "destroyed" arrays in the response are filtered according to the specified criteria. Clients using these filters MUST track the returned state and use it in subsequent Metadata/changes calls with the same filter parameters to ensure consistent synchronization.

If both filterRelatedType and filterMetadataType are specified, the server MUST return only changes to Metadata objects that satisfy both criteria (i.e., the filters are combined with a logical AND).

3.4. Metadata/query

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

3.4.1. Filtering

A *FilterCondition* object has the following properties, any of which may be omitted:

@type: String[]
: Only Metadata objects whose @type property value is in the specified array are returned. This allows filtering for specific metadata types, for example, to retrieve only Annotation objects or only ImapMetadata objects.

relatedType: String
: Only Metadata objects whose relatedType is equal to the specified value are returned. This is useful for retrieving metadata associated with a particular JMAP data type, such as all metadata on Email objects.

relatedIds: Id[]
: Only Metadata objects whose relatedId is in the specified array are returned. This allows retrieving metadata for a specific set of related objects. This property MUST only be specified when relatedType is also specified in the same FilterCondition object. If relatedIds is specified without relatedType, the server MUST reject the query with an "invalidArguments" error indicating that relatedType is required when using relatedIds.

isPrivate: Boolean
: Only Metadata objects whose isPrivate property matches the specified value are returned. This allows filtering to retrieve only private metadata (true) or only shared metadata (false).

textMatch: String
Only Metadata objects whose vendor-specific string properties contain the specified text are returned. The match SHOULD be case-insensitive and SHOULD look for the text anywhere within the property values. Servers MAY extend this to match against standard properties as well, but MUST at minimum search vendor-specific properties. The exact matching algorithm is implementation-defined, but servers SHOULD document their behavior for this filter.

3.4.2. Sorting

The following properties MUST be supported for sorting:

* id

The following properties SHOULD be supported for sorting:

* @type

* relatedType

* relatedId

* isPrivate

3.5. Metadata/queryChanges

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

4. Standard Method Extensions

This specification extends the standard JMAP methods for existing data types to support metadata operations. These extensions allow clients to retrieve and manage metadata alongside the primary data objects in a unified, efficient manner.

4.1. /get

This extension enhances the standard "/get" method defined in Section 5.1 of [RFC8620] for all JMAP data types specified in the dataTypes capability. The extension adds metadata retrieval capabilities to any supported data type's /get method (e.g., Email/get, Mailbox/get, Contact/get).

The following additional arguments are defined for the extended /get method:

fetchMetadata: Boolean (default: false)
If true, the server will return a metadata property in the response containing Metadata objects associated with the objects returned in the "list" property. If false, the metadata property will not be returned in the response. Servers MUST support this argument for all data types listed in the dataTypes capability.

metadataTypes: String[]|null (default: null)
This argument is only applicable when fetchMetadata is true. A list of metadata object type identifiers (@type values) to retrieve for the requested objects. If null, all metadata types are returned. If one or more type identifiers are specified, the server returns only metadata objects of those types associated with the requested objects. This argument is ignored if fetchMetadata is false. Servers MUST support this argument for all data types listed in the dataTypes capability.

metadataProperties: String[]|null (default: null)
This argument is only applicable when fetchMetadata is true. A list of metadata property names to include in the returned metadata objects. If null, all properties of the metadata objects are returned. If specified, only the requested properties are included in the response. The @type and relatedId properties MUST always be included in metadata objects even if not explicitly requested, as they are necessary to correlate metadata with their related objects and identify the metadata type. This argument follows the same semantics as the properties argument in the standard /get method. This argument is ignored if fetchMetadata is false. Servers MUST support this argument for all data types listed in the dataTypes capability.

The response to an extended /get method includes the following additional property when fetchMetadata is true:

metadata: Metadata[]
An array of Metadata objects associated with the objects returned in the list property. This property is included in the response if and only if the fetchMetadata argument was set to true in the request. The metadata array contains only those metadata objects whose relatedId corresponds to an object id in the list response. If metadataTypes was specified, only metadata objects whose @type matches one of the specified types are included. The relatedId property of each metadata object allows clients to correlate metadata with their related objects. If no metadata exists for the returned objects (or no metadata matches the specified type filters), this property MUST be an empty array.

If a requested related object does not have any metadata of the specified types, it simply will not have corresponding entries in the metadata array.

4.2. /set

This specification extends the standard "/set" method defined in Section 5.3 of [RFC8620] for all JMAP data types specified in the dataTypes capability. The extension adds metadata management capabilities to ensure proper lifecycle management and transactional consistency between primary objects and their associated metadata.

The following additional arguments are defined for the extended /set method:

onSuccessCreateMetadata: Id[Metadata[]]|null A map of object id to an array of Metadata objects to create, associated with the object referenced by the id, if the create/update of that object succeeds. (For references to objects created in the same "/set" invocation, this is equivalent to a creation-reference, so the id will be the creation id prefixed with a "#".)

The Metadata objects in the array MUST NOT include a relatedId or relatedType property; the server MUST automatically set relatedType to the data type of the /set method (e.g., "Email" for Email/set) and relatedId to the id of the successfully created or updated object. If any Metadata object specifies a relatedId or relatedType, the server MUST reject that metadata creation with an "invalidProperties" SetError.

onSuccessUpdateMetadata: Id[Id[PatchObject]]|null A map of object id to a map of Metadata object id to PatchObject containing properties to update on the Metadata object, if the create/update of the referenced object succeeds. (For references to objects created in the same "/set" invocation, this is equivalent to a creation-reference, so the id will be the creation id prefixed with a "#".)

The relatedId and relatedType properties MUST NOT be modified through this mechanism; attempts to do so MUST be rejected with an "invalidProperties" SetError. The specified Metadata objects MUST be associated with the corresponding related object; attempts to update metadata associated with other objects MUST be rejected with a "notFound" SetError.

If the onSuccessCreateMetadata or onSuccessUpdateMetadata arguments are specified and the corresponding object operations succeed, the server MUST perform an implicit Metadata/set call using the arguments

derived from these properties. The server MUST automatically populate relatedType and relatedId values for created Metadata objects based on the data type and id of the successfully processed object.

The response to the implicit Metadata/set call MUST be returned in an additional Metadata/set response immediately following the original /set response. This Metadata/set response MUST use the same method call id as the original /set request that triggered it.

If the original /set request fails entirely (e.g., due to an invalid accountId), no implicit Metadata/set call is made and no Metadata/set response is returned. If individual object operations within the /set request fail, the metadata operations associated with those failed objects are simply not attempted and do not appear in the Metadata/set response.

4.2.1. Cascading Deletion

When an object is destroyed through a /set method (e.g., Email/set with a destroy operation), the server MUST automatically delete all associated Metadata objects. This includes both private and shared metadata, regardless of which user created them. This cascading deletion ensures that metadata does not become orphaned when its related object is removed.

The automatic deletion of metadata applies to all metadata types (Annotation, ImapMetadata, WebDavMetadata, and any other registered metadata types) associated with the destroyed object.

Servers SHOULD NOT generate separate Metadata/set responses or state changes for automatically deleted metadata. The deletion is considered an implicit consequence of destroying the related object. However, servers MAY emit push notifications or update metadata state strings to reflect that metadata has been deleted, if such notifications would otherwise occur for explicitly deleted metadata.

Clients do not need to manually delete metadata before destroying related objects, and attempting to do so would be inefficient. The server handles metadata cleanup automatically.

If an error occurs during related object deletion that prevents the destruction from completing, the associated metadata MUST NOT be deleted either, maintaining referential integrity.

5. Access Control

Access control for Metadata objects is determined by the isPrivate property and the permissions on the parent object. This model ensures that metadata access aligns with the access control of the objects being annotated.

For private Metadata objects (isPrivate: true), only the authenticated user who created the Metadata object can read or modify it. Private metadata is isolated to the creating user and is not visible to other users regardless of their permissions on the parent object. This allows users to maintain personal annotations on shared objects without exposing them to collaborators or other users with shared access.

When processing Metadata/get or Metadata/query requests, the server MUST filter private Metadata objects to return only those created by the currently authenticated user. Private Metadata objects created by other users MUST NOT be included in the response, even if those other users have access to the same parent objects.

When processing Metadata/set requests for private Metadata objects, the server MUST verify that the user has appropriate access to the parent object (at minimum, read access to create private annotations on it). However, the user does not need write access to the parent object to create or modify their own private metadata about it.

For shared Metadata objects (isPrivate: false), access is governed by the permissions of the parent object. Users who have permission to read the parent object can also read shared Metadata objects associated with it. Users who have permission to modify the parent object can also modify shared Metadata objects associated with it.

For shareable JMAP object types as defined in Section 4 of [RFC9670] and type-specific specifications, the server MUST enforce access control based on the mayRead and mayWrite properties (or equivalent permission indicators) of the parent object when processing Metadata/get and Metadata/set requests on shared Metadata objects.

Specifically:

* A Metadata/get request for a shared Metadata object MUST only succeed if the requesting user has read permission (mayRead: true or equivalent) on the parent object.

* A Metadata/set request that creates or updates a shared Metadata object MUST only succeed if the requesting user has write permission (mayWrite: true or equivalent) on the parent object.

* A Metadata/set request that destroys a shared Metadata object MUST only succeed if the requesting user has write permission on the parent object (some servers may require additional permissions for deletion).

* For private Metadata objects, the server MUST verify that the requesting user has at least read access to the parent object before allowing creation, and that the user is the creator of the private Metadata object before allowing updates or deletion.

If a user attempts to access a Metadata object but lacks the necessary permissions, the server MUST reject the request with a "forbidden" error at the method level, or include a "forbidden" SetError in the notCreated/notUpdated/notDestroyed response for the specific object in Metadata/set operations.

If the maySetPrivate capability is false and a user attempts to create a private Metadata object, the server MUST reject the request with a "forbidden" SetError, regardless of their permissions on the parent object.

Servers SHOULD enforce these access control rules consistently across all Metadata operations. When a user’s permissions on a parent object change (for example, when sharing is revoked), the visibility of shared Metadata objects associated with that parent MUST change accordingly. Private Metadata objects remain accessible only to their creator regardless of permission changes on the parent object.

It is RECOMMENDED that servers provide clear error messages when access is denied, indicating whether the issue is lack of permission on the parent object, attempting to access another user’s private metadata, or lack of permission to create private metadata in the account.

6. Quota

Servers SHOULD enforce quota limits on the total storage consumed by Metadata objects within a JMAP account. Unbounded metadata storage could lead to resource exhaustion and denial of service conditions. Quota enforcement ensures fair resource allocation among users and protects server resources.

Servers that support the JMAP Quotas extension [RFC9425] SHOULD integrate metadata storage into the quota framework defined in that specification, allowing clients to query current metadata quota usage and limits through the standard Quota API. Servers that do not support [RFC9425] SHOULD still enforce implementation-defined limits on metadata storage and reject operations that would exceed those limits with an "overQuota" SetError.

The method for calculating the size of a Metadata object is implementation-specific, as different server architectures may have different storage characteristics. However, the following approach is RECOMMENDED for consistency and predictability:

Calculate the size of a Metadata object as the size of its CBOR [RFC8949] serialized representation in bytes. CBOR provides a compact binary encoding of structured data similar to JSON, and using CBOR size as the quota metric provides a consistent measure that is independent of the server's internal storage format. This approach accounts for all property names and values, including vendor-specific properties, in a standardized way.

When a Metadata/set operation would cause the account to exceed its metadata quota, the server MUST reject the operation with an "overQuota" SetError. The SetError SHOULD include a descriptive message indicating that the metadata quota has been exceeded. Servers MAY include additional information in the description, such as the current quota usage, the quota limit, and the size of the rejected operation.

Quota enforcement applies to both create and update operations. Creating a new Metadata object consumes quota, and updating an existing object may increase quota usage if the update adds or enlarges properties. Destroying a Metadata object MUST free the quota consumed by that object.

7. Examples

This section provides practical examples demonstrating common metadata management scenarios.

7.1. Creating a Mailbox with Annotation

This example shows how to create a new JMAP Mailbox and associate an Annotation object with vendor-specific properties in a single request.

```
[
  ["Mailbox/set", {
    "accountId": "A12345",
    "create": {
      "new-mailbox": {
        "name": "Project Alpha",
        "parentId": null,
        "role": null
      }
    }
  }, "c1"],
  ["Metadata/set", {
    "accountId": "A12345",
    "create": {
      "new-metadata": {
        "@type": "Annotation",
        "relatedType": "Mailbox",
        "relatedId": "#new-mailbox",
        "isPrivate": true,
        "acme.example.com:color": "blue",
        "acme.example.com:priority": "high",
        "acme.example.com:project": {
          "@type": "acme.example.com:ProjectInfo",
          "projectId": "ALPHA-2024",
          "deadline": "2024-12-31",
          "team": "Engineering"
        }
      }
    }
  }, "c2"]
]
```

7.2. Retrieving a Mailbox with Metadata

This example shows how to retrieve the created mailbox along with its associated Annotation metadata in a single request using the extended /get method:

Request:

```
[
  ["Mailbox/get", {
    "accountId": "A12345",
    "ids": ["MB789"],
    "properties": ["name", "parentId", "role"],
    "metadataTypes": ["Annotation"],
    "metadataProperties": ["acme.example.com:color",
                           "acme.example.com:priority"]
  }, "c3"]
]
```

Response:

```
[
  ["Mailbox/get", {
    "accountId": "A12345",
    "state": "m101",
    "list": [
      {
        "id": "MB789",
        "name": "Project Alpha",
        "parentId": null,
        "role": null
      }
    ],
    "metadata": [
      {
        "relatedId": "MB789",
        "@type": "Annotation",
        "acme.example.com:color": "blue",
        "acme.example.com:priority": "high"
      }
    ],
    "notFound": []
  }, "c3"]
]
```

7.3. Attaching Photography Metadata to a FileNode

This example demonstrates using a _fictional_ registered PhotoMetadata type to attach photography information to FileNode [I-D.ietf-jmap-filenode] objects. The PhotoMetadata type is assumed to be an IANA-registered metadata type specifically designed for photographic data.

Request:

```
[
  ["FileNode/set", {
    "accountId": "A12345",
    "update": {
      "F456": {
        "name": "lake-island.jpg"
      }
    }
  }, "c1"],
  ["Metadata/set", {
    "accountId": "A12345",
    "create": {
      "photo-meta": {
        "@type": "PhotoMetadata",
        "relatedType": "FileNode",
        "relatedId": "F456",
        "isPrivate": false,
        "geoLocation": {
          "latitude": 46.362,
          "longitude": 14.090
        },
        "cameraMake": "Canon",
        "cameraModel": "EOS R5",
        "aperture": "f/2.8",
        "shutterSpeed": "1/250",
        "iso": 400,
        "focalLength": "50mm",
        "dateTaken": "2023-10-01T01:14:00Z",
        "imageSize": {
          "width": 6000,
          "height": 4000
        }
      }
    }
  }, "c2"]
]
```

7.4. Retrieving a FileNode with Photography Metadata

Request:

```
[
  ["FileNode/get", {
    "accountId": "A12345",
    "ids": ["F456"],
    "metadataTypes": ["PhotoMetadata"],
    "metadataProperties": null
  }, "c3"]
]
```

Response:

```
[
  ["FileNode/get", {
    "accountId": "A12345",
    "state": "f201",
    "list": [
      {
        "id": "F456",
        "name": "lake-island.jpg",
        "type": "image/jpeg",
        "size": 2458624,
        "blobId": "Gabc123def456",
        "parentId": "folder789",
        "createdDate": "2024-07-15T20:00:00Z",
        "modifiedDate": "2024-10-20T15:30:00Z"
      }
    ],
    "metadata": [
      {
        "id": "photometa99",
        "@type": "PhotoMetadata",
        "relatedId": "F456",
        "geoLocation": {
          "latitude": 46.362,
          "longitude": 14.090
        },
        "cameraMake": "Canon",
        "cameraModel": "EOS R5",
        "aperture": "f/2.8",
        "shutterSpeed": "1/250",
        "iso": 400,
        "focalLength": "50mm",
        "dateTaken": "2023-10-01T01:14:00Z",
        "imageSize": {
          "width": 6000,
          "height": 4000
        }
      }
    ],
    "notFound": []
  }, "c3"]
]
```

7.5. Creating an Email with Annotation Atomically

This example demonstrates creating an Email and associating an Annotation with it in a single atomic operation using onSuccessCreateMetadata. The draft Email is created and, if successful, an Annotation is attached to track workflow state.

Request:

```
[
  ["Email/set", {
    "accountId": "A12345",
    "create": {
      "draft1": {
        "mailboxIds": { "MB123": true },
        "subject": "Project Update",
        "from": [{ "email": "alice@example.com" }],
        "to": [{ "email": "bob@example.com" }],
        "bodyStructure": {
          "type": "text/plain",
          "partId": "1"
        },
        "bodyValues": {
          "1": { "value": "Here is the project update..." }
        }
      }
    },
    "onSuccessCreateMetadata": {
      "#draft1": [
        {
          "@type": "Annotation",
          "isPrivate": true,
          "acme.example.com:workflowState": "pending-review",
          "acme.example.com:assignedTo": "carol@example.com"
        }
      ]
    }
  }, "c1"]
]
```

A successful response includes two responses with the same method call id, due to the implicit Metadata/set call:

```
[
  ["Email/set", {
    "accountId": "A12345",
    "oldState": "e100",
    "newState": "e101",
    "created": {
      "draft1": {
        "id": "EM456",
        "blobId": "Gxyz789",
        "threadId": "T001",
        "size": 1024
      }
    }
  }, "c1"],
  ["Metadata/set", {
    "accountId": "A12345",
    "oldState": "md50",
    "newState": "md51",
    "created": {
      "1": {
        "id": "MD789",
        "@type": "Annotation",
        "relatedType": "Email",
        "relatedId": "EM456"
      }
    }
  }, "c1"]
]
```

7.6. Updating a Calendar Event and its Metadata Atomically

This example demonstrates updating a CalendarEvent and its associated metadata atomically, ensuring that the metadata update only applies if the event update succeeds:

Request:

```
[
  ["CalendarEvent/set", {
    "accountId": "A12345",
    "update": {
      "CE789": {
        "title": "Quarterly Review Meeting",
        "start": "2024-12-15T14:00:00"
      }
    },
    "onSuccessUpdateMetadata": {
      "CE789": {
        "MD456": {
          "acme.example.com:lastModifiedReason":
              "Rescheduled per manager request",
          "acme.example.com:approvalStatus": "pending"
        }
      }
    }
  }, "c1"]
]
```

A successful response:

```
[
  ["CalendarEvent/set", {
    "accountId": "A12345",
    "oldState": "c200",
    "newState": "c201",
    "updated": {
      "CE789": null
    }
  }, "c1"],
  ["Metadata/set", {
    "accountId": "A12345",
    "oldState": "md60",
    "newState": "md61",
    "updated": {
      "MD456": null
    }
  }, "c1"]
]
```

8. Security considerations

The metadata management capabilities defined in this specification introduce several security considerations that implementers and deployers must address to protect user data and prevent abuse.

8.1. Metadata Confidentiality

Metadata objects may contain sensitive information, including personal notes, workflow states, application-specific data, and other user-generated content. Servers MUST enforce the access control mechanisms defined in Access Control (Section 5) to prevent unauthorized disclosure of metadata.

Private metadata (isPrivate: true) contains information intended only for the authenticated user who created it. Servers MUST ensure that private metadata is never exposed to other users through any interface, including JMAP, IMAP, WebDAV, or any other supported protocols. Each user's private metadata must remain isolated from all other users, even those who have full access to the parent object.

The per-user nature of private metadata introduces important implementation considerations. Since multiple users may each have their own private Metadata objects for the same parent object and metadata type, servers must carefully track which user created each private Metadata object. When processing queries or retrieval requests, servers MUST filter results to include only the private metadata belonging to the currently authenticated user. Failure to properly implement this filtering could result in serious privacy breaches where one user's private notes or annotations are exposed to other users.

Shared metadata visibility is tied to the permissions of the parent object. Servers MUST consistently enforce these permissions across all access methods. When sharing permissions on a parent object change, the visibility of associated shared metadata MUST change accordingly. Servers should be cautious about caching metadata in ways that could bypass updated permission checks.

For servers that bridge JMAP metadata to IMAP METADATA or WebDAV properties, special care must be taken to ensure that the visibility and access control semantics are correctly mapped between protocols. IMAP private metadata (entries under "/private/") must map to JMAP private metadata for the authenticated user. Access control on WebDAV properties must align with JMAP shared metadata permissions. Servers must ensure that the per-user isolation model is maintained when metadata is accessed through different protocol interfaces.

8.2. User Identity and Authentication

The private metadata model relies critically on accurate user authentication and identity management. Servers MUST:

* Reliably identify the authenticated user for all metadata operations

* Maintain accurate association between private Metadata objects and their creating users

* Prevent authentication bypass or identity spoofing that could allow access to other users' private metadata

* Handle account delegation or administrative access carefully to ensure private metadata is not inadvertently exposed (administrators accessing an account on behalf of a user should not see other users' private metadata on shared objects)

Servers that support account delegation, impersonation, or administrative access features must carefully consider whether delegated access should include visibility into the account owner's private metadata, and should provide controls to limit such access when appropriate.

8.3. Injection Attacks Through Vendor-Specific Properties

The extensibility mechanism that allows vendor-specific properties introduces potential security risks if not properly handled. Malicious clients could attempt to inject harmful content through vendor-specific properties, including:

* Script injection: Embedding executable code in metadata properties that might be rendered in a web context without proper sanitization

* XML/JSON injection: Crafting property values that could break parsing or processing logic

* Path traversal: Using specially crafted property names to attempt unauthorized file system access

* SQL injection: If metadata is stored in SQL databases, unsanitized property values could lead to SQL injection vulnerabilities

Servers MUST treat all vendor-specific property values as untrusted user input. When storing, processing, or displaying metadata, servers and clients MUST:

* Validate property names conform to the specified format (domain-name prefix followed by colon and property name)

* Sanitize property values before use in any context where interpretation could occur (HTML rendering, script execution, database queries, etc.)

* Apply appropriate output encoding when displaying metadata to users

* Enforce length limits on property names and values to prevent resource exhaustion

* Reject or sanitize properties containing control characters or other potentially harmful content

For WebDavMetadata objects, special attention must be paid to the serialized XML content in property values. Servers MUST:

* Validate that XML content is well-formed before storage

* Sanitize XML to remove potentially harmful constructs (scripts, entity expansions, external entity references)

* Apply appropriate XML entity encoding when processing or displaying the content

* Implement protections against XML external entity (XXE) attacks and XML bomb attacks

Clients that display metadata to users MUST treat metadata content as untrusted and apply appropriate security measures, such as Content Security Policy restrictions, HTML sanitization, and context-appropriate escaping.

8.4. Resource Exhaustion and Denial of Service

Without proper controls, metadata management features could be abused to cause resource exhaustion or denial of service:

* Storage exhaustion: Attackers could create excessive metadata to consume storage quota or server resources. The per-user private metadata model amplifies this risk since multiple users could each create private metadata on the same shared objects.

* Processing exhaustion: Complex nested vendor-specific properties or deeply nested object structures could consume excessive CPU or memory during processing

* Query complexity: Expensive metadata queries, particularly those involving private metadata filtering across many users, could strain server resources

Servers SHOULD implement the quota mechanisms described in Section 6 to limit total metadata storage per account. Additionally, servers SHOULD:

* Enforce the maxDepth limit on nested vendor-specific properties

* Implement timeouts for metadata operations to prevent long-running queries

* Apply rate limiting to metadata creation and modification operations

* Monitor for unusual patterns of metadata usage that might indicate abuse (e.g., one user creating private metadata on unusually large numbers of objects)

* Implement reasonable limits on the size of individual property values

* Consider implementing separate quota tracking for private vs. shared metadata to prevent abuse

Servers MAY reject metadata creation or updates that appear abusive, returning appropriate SetError responses.

8.5. Protocol Bridging Risks

For servers that bridge JMAP metadata to IMAP METADATA or WebDAV properties, inconsistencies or vulnerabilities in the bridging logic could lead to security issues:

* Permission bypass: Differences in permission models between protocols could be exploited to gain unauthorized access. For example, IMAP and WebDAV may have different concepts of private vs. shared metadata that must be carefully reconciled.

* Metadata corruption: Improper format conversion could corrupt metadata or enable injection attacks. Special care is needed when converting between JMAP's JSON representation, IMAP's string values, and WebDAV's XML properties.

* Information leakage: Protocol-specific metadata might inadvertently be exposed through the bridging mechanism. For instance, internal IMAP server entries or WebDAV live properties should not be exposed as JMAP metadata.

* Synchronization inconsistencies: Race conditions or improper locking when synchronizing metadata across protocol boundaries could lead to data corruption or access control violations.

Servers that implement protocol bridging MUST carefully validate the security properties of their bridging implementation and ensure that the most restrictive applicable access control is enforced. The per-user isolation model for private metadata must be maintained consistently across all protocol interfaces.

For read-only ImapMetadata and WebDavMetadata access, servers should clearly document this limitation to prevent client confusion and ensure that clients do not assume modification capabilities that are not actually available.

8.6. Server Vulnerabilities

Implementations must be robust against malformed requests and unexpected data. Servers MUST:

* Validate all input according to the type specifications in this document

* Handle JSON/XML parsing errors gracefully without exposing server internals

* Protect against integer overflow, buffer overflow, and other common vulnerabilities when processing metadata

* Properly handle the per-user filtering of private metadata to prevent information leakage

* Implement proper database indexing and query optimization to handle private metadata filtering efficiently even with large numbers of users and objects

* Log security-relevant events (authentication failures, authorization failures, suspicious patterns, attempts to access other users' private metadata) for monitoring and incident response

8.7. Client Vulnerabilities

Clients that implement metadata management must also consider security:

* Clients MUST validate server responses and handle unexpected or malformed data gracefully

* Clients SHOULD warn users before displaying potentially sensitive metadata, particularly when displaying shared metadata that might have been created by other users

* Clients MUST NOT execute or interpret metadata content as code unless the user explicitly requests it and appropriate sandboxing is in place

* Clients accessing metadata through web browsers should utilize security features such as Content Security Policy

* Clients should clearly distinguish between private and shared metadata in the user interface to prevent users from inadvertently sharing private information

* When parsing XML content from WebDavMetadata properties, clients must use secure XML parsers and disable dangerous features like external entity resolution

9. IANA considerations

9.1. JMAP Capability Registration for "metadata"

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

Capability Name: urn:ietf:params:jmap:metadata
Specification document: this document
Intended use: common
Change Controller: IETF
Security and privacy considerations: this document, Section 8

9.2. JMAP Data Type Registration for "Metadata"

IANA will register "Metadata" in the "JMAP Data Types" registry as follows:

Type Name: Metadata
Can reference blobs: no
Can Use for State Change: yes
Capability: urn:ietf:params:jmap:metadata
Specification document: this document

9.3. Creation of the "JMAP Metadata Properties" Registry

IANA has created the "JMAP Metadata Properties" registry to allow interoperability of extensions to JMAP Metadata objects.

This registry follows the Expert Review process ([RFC8126], Section 4.5). If the "Intended Usage" field is common, sufficient documentation is required to enable interoperability. Preliminary community review for this registry is optional but strongly encouraged.

A registration can have an intended usage of common, reserved, or obsolete. IANA will list registrations with a common usage designation prominently and separately from those with other intended usage values.

A reserved registration reserves a property name without assigning semantics to avoid name collisions with future extensions or protocol use.

An obsolete registration denotes a property that is no longer expected to be added by up-to-date systems. A new property has probably been defined covering the obsolete property’s semantics.

The JMAP Metadata property registration procedure is not a formal standards process but rather an administrative procedure intended to allow community comment and check it is coherent without excessive time delay. It is designed to encourage vendors to document and register new properties they add for use cases not covered by the original specification, leading to increased interoperability

9.3.1. Preliminary Community Review

Notice of a potential new registration SHOULD be sent to the JMAP mailing list jmap@ietf.org (mailto:jmap@ietf.org) for review. This mailing list is appropriate to solicit community feedback on a proposed new property.

Property registrations must be marked with their intended use: "common", "reserved", or "obsolete".

The intent of the public posting to this list is to solicit comments and feedback on the choice of the property name, the unambiguity of the specification document, and a review of any interoperability or security considerations. The submitter may submit a revised registration proposal or abandon the registration completely at any time.

9.3.2. Change Procedures

Once a JMAP Metadata property has been published by IANA, the change controller may request a change to its definition. The same procedure that would be appropriate for the original registration request is used to process a change request.

JMAP Metadata property registrations may not be deleted; properties that are no longer believed appropriate for use can be declared obsolete by a change to their "intended usage" field; such properties will be clearly marked in the IANA registry.

Significant changes to a JMAP Metadata property’s definition should be requested only when there are serious omissions or errors in the published specification, as such changes may cause interoperability issues. When review is required, a change request may be denied if it renders entities that were valid under the previous definition invalid under the new definition.

The owner of a JMAP Metadata property may pass responsibility to another person or agency by informing IANA; this can be done without discussion or review.

9.3.3. "JMAP Metadata Properties" Registry Template

Property Name:
: This is the name of the property. The property name MUST NOT already be registered for any of the object types listed in the "Property Context" field of this registration. Other object types MAY already have registered a different property with the same name; however, the same name SHOULD only be used when the semantics are analogous.

Property Type:
: This is the type of this property, using type signatures, as specified in Section 2.2.1.1. The property type MUST be registered in the "JMAP Metadata Types" registry.

Property Context:
: This is a comma-separated list of JMAP Metadata object types this property is allowed on.

Reference or Description:
This is a brief description or RFC number and section reference where the property is specified (omitted for "reserved" property names).

Intended Usage:
This may be "common", "reserved", or "obsolete".

Change Controller:
This is who may request a change to this entry’s definition (IETF for RFCs from the IETF stream).

9.3.4. Submit Request to IANA

Registration requests can be sent to iana@iana.org (mailto:iana@iana.org).

9.3.5. Designated Expert Review

The primary concern of the designated expert (DE) is preventing name collisions and encouraging the submitter to document security and privacy considerations. For a common-use registration, the DE is expected to confirm that suitable documentation, as described in Section 4.6 of [RFC8126], is available to ensure interoperability. That documentation will usually be in an RFC, but simple definitions are likely to use a web/wiki page, and if a sentence or two is deemed sufficient, it could be described in the registry itself. The DE should also verify that the property name does not conflict with work that is active or already published within the IETF. A published specification is not required for reserved or obsolete registrations.

The DE will either approve or deny the registration request and publish a notice of the decision to the JMAP WG mailing list or its successor, as well as inform IANA. A denial notice must be justified by an explanation, and, in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable should be provided.

9.3.6. Initial Contents for the "JMAP Metadata Properties" Registry

The following table lists the initial entries of the "JMAP Metadata Properties" registry. All properties are for common use. All RFC section references are for this document. The change controller for all these properties is "IETF".

Property Name	Property Type	Property Context	Reference or Description
@type	String	Annotation, ImapMetadata, WebDavMetadata	Section 2.2.1.1
id	Id	Annotation, ImapMetadata, WebDavMetadata	Section 2.2.1.2
relatedType	String	Annotation, ImapMetadata, WebDavMetadata	Section 2.2.1.3
relatedId	Id	Annotation, ImapMetadata, WebDavMetadata	Section 2.2.1.4
isPrivate	Boolean	Annotation, ImapMetadata, WebDavMetadata	Section 2.2.1.5
metadata	[String]String	ImapMetadata, WebDavMetadata	Section 2.2.2.1, Section 2.2.3.1

Table 1: Initial Contents of the "JMAP Metadata Properties" Registry

9.4. Creation of the "JMAP Metadata Types" Registry

IANA has created the "JMAP Metadata Types" registry to avoid name collisions and provide a complete reference for all metadata types used for JMAP Metadata property values. The registration process is the same as for the "JMAP Metadata Properties" registry, as defined in Section 9.3.

9.4.1. "JMAP Metadata Types" Registry Template

Type Name:
the name of the type

Applicable Data Types:

a list of JMAP data type names that this metadata type can be associated with, or "any" if the metadata type can be associated with any JMAP data type

Reference or Description:
a brief description or RFC number and section reference where the Type is specified (may be omitted for "reserved" type names)

Intended Use:
common, reserved, or obsolete

Change Controller:
who may request a change to this entry’s definition (IETF for RFCs from the IETF stream)

9.4.2. Initial Contents for the "JMAP Metadata Types" Registry

The following table lists the initial entries of the JMAP Metadata Types registry. All properties are for common use. All RFC section references are for this document. The change controller for all these properties is "IETF".

Type Name	Applicable Data Types	Reference or Description
Annotation	any	Section 2.1.1
ImapMetadata	Mailbox	Section 2.1.2
WebDavMetadata	Calendar, CalendarEvent, AddressBook, ContactCard, FileNode	Section 2.1.3

Table 2: Initial Contents of the "JMAP Metadata Types" Registry

10. References

10.1. Normative References

[I-D.ietf-jmap-calendars]
Jenkins, N. and M. Douglass, "JSON Meta Application Protocol (JMAP) for Calendars", Work in Progress, Internet-Draft, draft-ietf-jmap-calendars-26, 4 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-jmap-calendars-26>.

[I-D.ietf-jmap-filenode]
Gondwana, B., "JMAP File Storage extension", Work in Progress, Internet-Draft, draft-ietf-jmap-filenode-03, 17 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-jmap-filenode-03>.

[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/rfc/rfc2119>.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <https://www.rfc-editor.org/rfc/rfc4918>.

[RFC5464] Daboo, C., "The IMAP METADATA Extension", RFC 5464, DOI 10.17487/RFC5464, February 2009, <https://www.rfc-editor.org/rfc/rfc5464>.

[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/rfc/rfc8126>.

[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/rfc/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/rfc/rfc8620>.

[RFC9610] Jenkins, N., Ed., "JSON Meta Application Protocol (JMAP) for Contacts", RFC 9610, DOI 10.17487/RFC9610, December 2024, <https://www.rfc-editor.org/rfc/rfc9610>.

[RFC9670] Jenkins, N., Ed., "JSON Meta Application Protocol (JMAP) Sharing", RFC 9670, DOI 10.17487/RFC9670, November 2024, <https://www.rfc-editor.org/rfc/rfc9670>.

10.2. Informative References

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <https://www.rfc-editor.org/rfc/rfc3501>.

[RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <https://www.rfc-editor.org/rfc/rfc4791>.

[RFC6352] Daboo, C., "CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)", RFC 6352, DOI 10.17487/RFC6352, August 2011, <https://www.rfc-editor.org/rfc/rfc6352>.

[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/rfc/rfc8621>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9425] Cordier, R., Ed., "JSON Meta Application Protocol (JMAP) for Quotas", RFC 9425, DOI 10.17487/RFC9425, June 2023, <https://www.rfc-editor.org/rfc/rfc9425>.

Appendix A. Changes

[[This section to be removed by RFC Editor]]

draft-ietf-jmap-metadata-01

* Renamed parentType and parentId properties to relatedType and relatedId throughout the document to better reflect their purpose and avoid implying a strict hierarchy.

* Added filterRelatedType and filterMetadataType optional parameters to Metadata/changes method to allow filtering changes by related object type and metadata type.

* Changed the error type from "invalidProperties" to "alreadyExists" when a client attempts to create a Metadata object that violates the uniqueness constraint.

* Added requirement that relatedType must be specified in the same FilterCondition object whenever relatedIds is used in Metadata/query.

* Added id as a MUST support property for sorting in Metadata/query; other properties changed to SHOULD support.

* Replaced the metadataTypes argument with a fetchMetadata Boolean argument (default: false) in the extended /get method.

* Added onSuccessCreateMetadata and onSuccessUpdateMetadata arguments to the extended /set method to enable transactional metadata operations that are conditional on the success of the related object operation.

draft-ietf-jmap-metadata-00

* Initial version

Author's Address

Mauro De Gennaro
Stalwart Labs LLC
1309 Coffeen Avenue, Suite 1200
Sheridan, WY 82801
United States of America
Email: mauro@stalw.art
URI: https://stalw.art