Network Working Group
Internet-Draft
Updates: 8620 (if approved)
Intended status: Standards Track
Expires: 21 August 2026

B. Gondwana
Fastmail

17 February 2026

JMAP File Storage extension
draft-ietf-jmap-filenode-06

Abstract

The JMAP base protocol (RFC8620) provides the ability to upload and download arbitrary binary data. This binary data is called a "blob", and can be used in all other JMAP extensions.

This extension adds a method to expose blobs as a filesystem along with the types of metadata that are provided by other remote filesystem protocols.

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 21 August 2026.

Copyright Notice

Table of Contents

1. Introduction

JMAP ([JMAP-CORE] — JSON Meta Application Protocol) is a generic protocol for synchronizing data 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.

In the same way that JMAP Calendars ([JMAP-CALENDARS]) replaces CalDAV ([CALDAV]) and JMAP Contacts ([JMAP-CONTACTS]) replaces CardDAV ([CARDDAV]), this document replaces the use of WebDAV ([WEBDAV]) for remote filesystem access.

2. Addition to the Capabilities Object

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

This document defines an additional capability URI.

2.1. urn:ietf:params:jmap:filenode

The capability urn:ietf:params:jmap:filenode being present in the "accountCapabilities" property of an account represents support for the FileNode datatype. Servers that include the capability in one or more "accountCapabilities" properties MUST also include the property in the "capabilities" property.

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

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

* maxFileNodeDepth: "UnsignedInt|null"

  The maximum depth of the FileNode hierarchy (i.e., one more than the maximum number of ancestors a FileNode may have), or null for no limit.

* maxSizeFileNodeName: "UnsignedInt"

  The maximum length, in (UTF-8) octets, allowed for the name of a FileNode. This MUST be at least 100, although it is recommended servers allow more.

* fileNodeQuerySortOptions: "String[]"

  A list of all the values the server supports for the "property" field of the Comparator object in an "FileNode/query" sort (see Section XXX). This MAY include properties the client does not recognise (for example, custom properties specified in a vendor extension). Clients MUST ignore any unknown properties in the list.

* mayCreateTopLevelFileNode: "Boolean"

  If true, the user may create a FileNode (see Section XXX) in this account with a null parentId. (Permission for creating a child of an existing FileNode is given by the "myRights" property on that FileNode.)

2.1.1. Capability Example

TODO

3. FileNode Data Type

A FileNode is a set of metadata which behaves similar to an inode in a filesystem. In [WEBDAV] terminology a FileNode can refer to either a collection or a resource.

The following JMAP Methods are selected by the urn:ietf:params:jmap:filenode capability.

3.1. FileNode objects

The filenode object has the following keys:

* id: "Id" (immutable; server-set)

  The Id of the FileNode.

* parentId: "Id|null"

  The Id of the parent node, or null if this is a top level node.

* blobId: "Id|null"

  The blobId for the content of this node, or null if this node is a collection. NOTE the zero byte file MUST have a non-null blobId. The blobId is immutable after creation.

* size: "UnsignedInt|null" (server-set)

  The size in bytes of the associated blob data. This must be null if, and only if, the blobId is null. Size is optional on create, but if provided it MUST match the size of the provided blobId, or be null if the node is a directory. The size is immutable after creation.

* name: "String"

User-visible name for the FileNode. This MUST be a Net-Unicode string [UNICODE] of at least 1 character in length, subject to the maximum size given in the capability object. There MUST NOT be two sibling Mailboxes with both the same parent and the same name. Servers MAY reject names that violate server policy (e.g., names containing control characters). Further:

- The name MUST NOT be "." or ".."

- The name MUST NOT contain a "/"

TODO: are there other characters which we should forbid (e.g. ':' is not allowed on Windows)

A server MUST order creation and deletion operations within a single FileNode/set such that the sibling constraint is retained at the end of the transaction, but replacing an existing file can be done atomically.

* type: "String|null"

  The media type of the FileNode. This MUST be null if, and only if, the node does not have a blobId.

  Valid values are found in the IANA media-types registry.

  Servers MUST NOT reject media types that are not recognised.

  Servers MUST reject media types if the value does not conform to the ABNF of [MEDIATYPE] Section 4.2.

  The type is immutable after creation.

* created: "UTCDate" (default: current server time)

  The date the node was created.

* modified: "UTCDate" (default: current server time)

  The date the node was last updated. NOTE: this is not updated by the server, clients must store a new value when making changes.

* accessed: "UTCDate" (default: current server time)

  The date the node was last accessed. NOTE: this is not updated by the server, clients must store a new value. See Implementation Considerations for comments on the use of this field.

* executable: "Boolean" (default: false)

  If true, the node is should be treated as an executable by operating systems that support this flag.

* isSubscribed: "Boolean" (default: true)

  This property is stored per user, and if true, the node should be displayed to the current user. Some servers may not allow this field to be changed for some or all nodes, e.g. only for directories, or only for nodes on which the shareWith ACL is actually set for this user, not nodes which inheret their ACL from a parent.

* myRights: "FilesRights" (server-set)

  The set of rights (ACLs) the user has in relation to this folder. A *FilesRights* object has the following properties:

  - mayRead: Boolean The user may read the contents of this node.

  - mayWrite: Boolean The user may modify the properties of this node, including renaming or deleting children.

  - mayShare: Boolean The user may change the sharing of this node (see [JMAP-SHARING])

* shareWith: "Id[FilesRights]|null"

  A map of userId to rights for users this node is shared with. The owner of the node MUST NOT be in this set. This is null if the user requesting the object does not have myRights.mayShare, or if the node is not shared with anyone.

* role: "String|null"

  An indication that this directory has a special role. The role MUST be null for files.

  TODO: we'll need to decide if there's an existin registry or whether we need a registry for known roles.

  The initial known roles are:

  - "root" - the base of a filesystem.

  - "home" - a user's home directory.

- "temp" - a temporary space - things stored in here are expected to be cleaned up automatically

- "trash" - a place where deleted data is moved to, it's expected that data from here will be deleted either automatically or manually.

3.2. FileNode Methods

3.2.1. FileNode/set

This is a standard Foo/set method, with the following differences:

Since parentId creates a tree structure, an attempt to move a node to a parent for which this node is also an ancestor is an error, and an invalidProperties error will be returned.

There are these additional top level arguments:

* onDestroyRemoveChildren: "Boolean" (default: false)

  If false, an attempt to destroy a FileNode which is the parentId of another FileNode will be rejected with a nodeHasChildren error. NOTE: if all the child nodes are being destroyed in the same operation, then the server MUST NOT return this error. Servers must either sort the destroys children before parents, or only check this constraint on the final state, remembering that JMAP set operations must be atomic.

  If true, then all child nodes will also be destroyed when a node is destroyed. When deleting child nodes, the server MUST include the ids of all deleted nodes in the method response.

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

  If null, an attempt to create or update a FileNode which would cause a name collision will be rejected by the server with a "alreadyExists" error.

  If "replace", the existing item will destroyed. In this case, the server MUST include the id of the replaced item in the destroyed response list. NOTE: if the replaced item is a directory which contains folders, then the server MUST respond with a nodeHasChildren error to this action unless onDestroyRemoveChildren is true.

If "rename", the server will change the "name" field to not clash, using an algorithm of its choice. If the server changes the name, it MUST include the new "name" value in the created or updated response field for this id.

3.2.2. FileNode/copy

This is a standard Foo/copy function with the same additional top-level arguments as FileNode/set, onDestroyRemoveChildren and onExists, with the same behaviour.

3.2.3. FileNode/get

This is a standard Foo/get method.

3.2.4. FileNode/changes

This is a standard Foo/changes method.

3.2.5. FileNode/query

This is a standard Foo/query method except for the following:

There’s one more property to the query:

* depth: "UnsignedInt|null"

  The number of levels of subdiretories to recurse into. If absent, null, or zero, do not recurse.

The following filter criteria are defined:

* isTopLevel: "Boolean"

  If true, the node must have a null parentId to match the condition.

* parentId: "Id"

  A FileNode id. A node must have a parentId equal to this to match the condition.

* ancestorId: "Id"

  A FileNode id. A node must have an ancestor (parent, parent of parent, etc.) with an id equal to this to to match the condition.

* hasType: "Boolean"

If true, the FileNode must be a file/resource, not a directory/ collection.

* hasRole: "String"

  A role name. Only nodes with precisely this role match this condition.

* hasAnyRole: "Boolean"

  If true, any node with a defined role matches this condition. If false, any node which has a role does not match this condition.

* blobId: "Id"

  A FileNode must have a blobId equal to this to match the condition.

* isExecutable: "Boolean"

  If true, the FileNode must have a true executable value.

* createdBefore: "UTCDate"

  The creation date of the node (as returned on the FileNode object) must be before this date to match the condition.

* createdAfter: "UTCDate"

  The creation date of the node (as returned on the FileNode object) must be on or after this date to match the condition.

* modifiedBefore: "UTCDate"

  The modified date of the node (as returned on the FileNode object) must be before this date to match the condition.

* modifiedAfter: "UTCDate"

  The modified date of the node (as returned on the FileNode object) must be on or after this date to match the condition.

* accessedBefore: "UTCDate"

  The accessed date of the node (as returned on the FileNode object) must be before this date to match the condition.

* accessedAfter: "UTCDate"

The accessed date of the node (as returned on the FileNode object) must be on or after this date to match the condition.

* minSize: "UnsignedInt"

  The size of the node in bytes (as returned on the FileNode object) must be equal to or greater than this number to match the condition.

* maxSize: "UnsignedInt"

  The size of the node in bytes (as returned on the FileNode object) must be less than this number to match the condition.

* name: "String"

  A FileNode must have exactly the same octets in its name property to match the condition.

* nameMatch: "String"

  Does a glob match of the specified name against the _name_ property of the node.

* type: "String"

  A FileNode must have exactly the same octets in its type property to match the condition

* typeMatch: "String"

  Does a glob match of the specified type against the _type_ property of the node.

* body: "String"

  Match the content of the referenced blob, see the definition of _body_ in section 4.4.1 of [JMAP-MAIL]. The server may use any technology to extract meaningful text from the blob for searching, or interpret the string in any way, to choose the nodes that it believes the user wants to see.

* text: "String"

  Is equivalent to _body_ OR _nameMatch_ OR _typeMatch_.

It also supports the following additional sort properties:

* tree:

  Sort by tree; which means by name, but any directory/collection node is immediately followed by the recursive application of the same sort to its child nodes. This is similar to the output of the find command on a filesystem with the depth parameter provided above.

* hasType:

  Sort directories before files (false sorts before true)

* type:

  Sorts directories first, and sorts by media type for files

3.2.6. FileNode/queryChanges

This is a standard Foo/queryChanges method.

4. Access Control

Since nodes create a tree, ACLs created by shareWith automatically apply to children as well, so if mayRead is set on a node, all its child nodes are also readable.

If a server does not support changing access on non-directory nodes, it can set mayShare to false on those nodes, even if the parent directory has true.

Nodes which are not "discoverable" MUST return notFound errors if fetched with FileNode/get and MUST NOT be returned in response to FileNode/query. Nodes are discoverable in one of two ways:

1. If the Node or an ancestor of the Node has mayRead true.

2. If the Node is an ancestor of a Node which has mayRead true.

In the second case, the Node itself will have mayRead false, and appears only to give the full path to the visible Nodes. This means that a query with that Node as the parent will only return those of its children which are otherwise discoverable through the second discoverability rule.

This leads to a sharee seeing the full path to anything that they have access to, but nothing else.

E.g. in a unix homedirectory environment where user Alice had shared the "forBob" folder, one might see, when logged in as Bob:

```
/home
  /alice
    /shared
      /forBob
        file1.jpg
        file2.txt
        ...
  /bob
    bobfile.txt
    ...
```

While Alice would see many more files and folders at the home directory level.

This does lead to potentially large changes in the visible Node set, so when Alice first shared that directory, Bob would have been told of a large number of "created" Nodes in response to a FileNode/ changes query.

5. Security considerations

TODO: lots of "filesystems are risky" - I guess look at the referenced RFCs and what they said.

6. IANA considerations

6.1. JMAP Capability registration for "filenode"

IANA is requested to register the "filenode" JMAP Capability as follows:

Capability Name: urn:ietf:params:jmap:filenode

Specification document: this document

Intended use: common

Change Controller: IETF

Security and privacy considerations: this document, Section XXX

6.2. JMAP Error Codes registration for "nodeHasChildren"

IANA is requested to register the "nodeHasChildren" JMAP Error Code as follows:

JMAP Error Code: nodeHasChildren

Intended use: common

Change Controller: IETF

Description: The node being destroyed is still referenced by other nodes which have not been destroyed.

Reference: this document

6.3. JMAP Data Types registration for "FileNode"

IANA is requested to register the "FileNode" JMAP Data Type as follows:

Type Name: FileNode

Can Reference Blobs: Yes

Can Use For State Change: Yes

Capability: urn:ietf:params:jmap:filenode

Reference: this document

7. TODO

* support SYMLINK types (RFC4437)
* design and document the capabilities object
* create real-world clients to test this
* security considerations
* a way to get or query all ancestor nodes
* QUESTION: should all the file-related fields be embedded in a sub-object? There's lots of "must be NULL if and only-if this other field is also NULL" - we could enforce that more easily with a sub-object.
* We need to address how shareWith and myRights expiration are done; because both a potential fullPath and the real myRights depend on changes to parent nodes.

8. Changes

EDITOR: please remove this section before publication.

The source of this document exists on github at:
https://github.com/brong/draft-gondwana-jmap-filenode/

draft-ietf-jmap-filenode-06

* Documented FileNode/copy and detailed that it has the same new top-level keys

draft-ietf-jmap-filenode-05

* Renamed onDuplicate to onExists for name alignment

* added "role" for directories

* added a "TODO" for putting more restrictions on node names

* changed hasParentId to isTopLevel with reversed boolean meaning

draft-ietf-jmap-filenode-04

* Documented that blobId, size, and type are immutable after creation.

* Documented that creating a file and deleting the old copy of the file within a transaction is legal.

* Added onDuplicate top-level option for FileNode/set, giving both "rename" and "replace" options.

* Updated the definition of shareWith to say that the keys of the hash are Ids, not arbitrary strings

draft-ietf-jmap-filenode-03

* Added ’text’ and ’body’ searches (added JMAP-MAIL reference as additional information for body search)

* Updated JMAP-CONTACTS and JMAP-SHARING references to published RFC numbers rather than draft names

* Noted that the server MUST included the nodeids of deleted child nodes.

* Added isSubscribed

* Renamed mayAdmin to mayShare to align with other specs

* Described the inheretence of ACLs

draft-ietf-jmap-filenode-02

* Convert to Kramdown-RFC format (no intentional changes)

draft-ietf-jmap-filenode-01

* Refreshing draft only

draft-ietf-jmap-filenode-00

* upload as a working group document

draft-gondwana-jmap-filenode-01

* require a blobId for the zero-byte file

* make size also null for collections

* add more to the TODO section

* bikeshed; FileNode

* correct UTCDate, UnsignedInt, and normalised UTF-8.

* add some fields to the capabilities object

draft-gondwana-jmap-filenode-00

* initial proposal

9. Acknowledgements

Neil Jenkins and the JMAP working group at the IETF.

{backmatter}

10. References

10.1. Normative References

[JMAP-CORE]
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>.

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

[MEDIATYPE]
Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

10.2. Informative References

[CALDAV] Desruisseaux, B., Ed., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 5545, DOI 10.17487/RFC5545, September 2009, <https://www.rfc-editor.org/rfc/rfc5545>.

[CARDDAV] 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>.

[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>.

[JMAP-CONTACTS]
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>.

[JMAP-MAIL]
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>.

[UNICODE] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/rfc/rfc5198>.

[WEBDAV] 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>.

Author’s Address

Bron Gondwana
Fastmail
Email: brong@fastmailteam.com