Network Working Group
Internet-Draft
Obsoletes: 9404 (if approved)
Updates: 8620 (if approved)
Intended status: Standards Track
Expires: 23 October 2026

B. Gondwana
Fastmail

21 April 2026

JMAP Blob Management
draft-ietf-jmap-blobext-01

Abstract

The JSON Meta Application Protocol (JMAP) base protocol ([JMAP-CORE]) provides the ability to upload and download arbitrary binary data via HTTP POST and GET on a defined endpoint. This binary data is called a "blob".

This extension adds additional ways to create and access blobs by making inline method calls within a standard JMAP request. It also adds a reverse lookup mechanism to discover where blobs are referenced within other data types, support for large blobs via chunked construction with server-side optimisation, and server-side blob conversion operations including image format conversion, archive creation and extraction, compression and decompression, and delta/patch operations.

Table of Contents

1. Introduction

Sometimes JMAP ([JMAP-CORE]) interactions require creating a blob and then referencing it. Embedding blobs directly into the JMAP method calls array can reduce round trips.

Likewise, when fetching an object, it can be useful to also fetch the raw content of that object without a separate round trip.

When JMAP is proxied through a system that applies additional access restrictions, it can be useful to know which objects reference any particular blob; this document defines a way to discover those references.

For large blobs, a client may wish to upload data in chunks and have the server assemble them efficiently, or retrieve information about how the server has stored a blob internally.

Many applications also benefit from server-side transformations — converting images, creating or unpacking archives, compressing or decompressing data, or computing and applying deltas — without needing to download and re-upload blob content.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 one additional capability URI.

2.1. urn:ietf:params:jmap:blob2

The capability urn:ietf:params:jmap:blob2 being present in the "accountCapabilities" property of an account represents support for the blob methods defined in this document: Blob/set, Blob/get, Blob/lookup, and Blob/convert. Servers that include this capability in one or more "accountCapabilities" properties MUST also include it in the "capabilities" property.

A server MAY also advertise urn:ietf:params:jmap:blob (as defined in RFC9404) to support older clients, but a client MUST NOT include both urn:ietf:params:jmap:blob and urn:ietf:params:jmap:blob2 in the using array of the same request. The blob2 capability supersedes the blob capability and does not depend on it.

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:

* maxSizeBlobSet: "UnsignedInt|null"

  The maximum size of a blob (in octets) that the server will allow to be created (including blobs created by concatenating multiple data sources together). If null, the server does not advertise a specific limit but MAY still reject blobs that are too large.

* maxDataSources: "UnsignedInt"

The maximum number of DataSourceObjects allowed in the data array of a single creation in Blob/set. Servers MUST support at least 64 DataSourceObjects per creation.

* supportedTypeNames: "String[]"

  An array of data type names that the server supports for Blob/lookup. The values are the names listed in the IANA "JMAP Data Types" registry. If the array is empty, the server does not support Blob/lookup. Servers MAY include private type names not in the registry; clients MUST ignore names they do not recognise. Clients MUST also ignore names that they do recognise if the corresponding capability for that type name is not present in the 'capabilities' section of the session object, because that means that the name has been used as a private type rather than a capability that the client knows.

* supportedDigestAlgorithms: "String[]"

  An array of digest algorithms supported for Blob/get, from the IANA "Hash Function Textual Names" registry, expressed in lowercase (e.g., "sha", "sha-256"). Clients SHOULD prefer algorithms listed earlier in the array.

* uploadUrl: "String|null"

  A URI template (optionally containing {accountId}) for uploading blobs via HTTP POST for this account, in the same format as the session-level uploadUrl defined in [JMAP-CORE], Section 2. Servers MUST still provide the session-level uploadUrl; this per-account URL is an alternative that MAY point to a different server or be more efficient for this account. Uploads to this URL return a JSON object with the same properties as the session-level upload response, plus an expires property as defined in "The expires Response Property" below. If null, clients SHOULD use the session-level uploadUrl.

* chunkSize: "UnsignedInt|null"

  A hint indicating the preferred chunk size in octets. If a client uploads blobs with exactly this size except for the final chunk, and uses Blob/set with DataSourceObjects referencing these chunks, the server can optimise storage of these chunks. Servers MUST allow other sizes for the individual data blocks in Blob/set though, and will then choose whether to store them as an array of blobs still, or to combine them.

* supportedImageReadTypes: "String[]|null"

The media types ([MEDIA-TYPES]) that the server can read as input to ImageConvertRecipe. If null, the server does not support image conversion.

* supportedImageWriteTypes: "String[]|null"

  The media types ([MEDIA-TYPES]) that the server can produce as output from ImageConvertRecipe. If null, the server does not support image conversion.

* supportedArchiveTypes: "String[]|null"

  The archive media types ([MEDIA-TYPES]) supported for creating archives via ArchiveRecipe. If null, the server does not support archive creation.

* supportedExtractTypes: "String[]|null"

  The archive MIME types supported for extracting archives via ExtractRecipe. This MAY include types not in supportedArchiveTypes (e.g., "application/vnd.ms-tnef"). If null, the server does not support archive extraction.

* supportedCompressTypes: "String[]|null"

  The compression media types ([MEDIA-TYPES]) supported for compressing via CompressRecipe. If null, the server does not support compression.

* supportedDecompressTypes: "String[]|null"

  The compression MIME types supported for decompressing via DecompressRecipe. This MAY include types not in supportedCompressTypes. If null, the server does not support decompression.

* supportedDeltaTypes: "String[]|null"

  The delta media types ([MEDIA-TYPES]) supported for computing deltas via DeltaRecipe. If null, the server does not support delta computation.

* supportedPatchTypes: "String[]|null"

  The delta media types supported for applying patches via PatchRecipe. This MAY include types not in supportedDeltaTypes. If null, the server does not support patch application.

* maxConvertSize: "UnsignedInt|null"

  If supplied, the maximum size in octets of any single input blob to a Blob/convert operation. Requests referencing a blob larger than this value MUST be rejected with a "tooLarge" SetError. If null, the server does not advertise a specific limit but MAY still reject blobs that are too large.

* maxArchiveEntries: "UnsignedInt|null"

  If supplied, the maximum number of entries allowed in an ArchiveRecipe. Requests exceeding this limit MUST be rejected with a "tooLarge" SetError. If null, the server does not advertise a specific limit but MAY still reject requests with too many entries.

* maxImageDimension: "UnsignedInt|null"

  If supplied, the maximum value accepted for width or height in an ImageConvertRecipe, in pixels. Requests exceeding this limit MUST be rejected with a "tooLarge" SetError. If null, the server does not advertise a specific limit but MAY still reject requests with dimensions that are too large.

2.1.1. Capability Example

```
{
  "urn:ietf:params:jmap:blob2": {
    "maxSizeBlobSet": 52428800,
    "maxDataSources": 64,
    "supportedTypeNames": ["Email", "Thread", "Mailbox"],
    "supportedDigestAlgorithms": ["sha-256", "sha"],
    "uploadUrl": "https://upload.example.com/jmap/upload/{accountId}/",
    "chunkSize": 5242880,
    "supportedImageReadTypes": [
      "image/png",
      "image/jpeg",
      "image/gif",
      "image/tiff"
    ],
    "supportedImageWriteTypes": [
      "image/png",
      "image/jpeg"
    ],
    "supportedArchiveTypes": [
      "application/zip",
      "application/x-tar"
    ],
```

```
      "supportedExtractTypes": [
        "application/zip",
        "application/x-tar",
        "application/vnd.ms-tnef"
      ],
      "supportedCompressTypes": [
        "application/gzip",
        "application/zstd"
      ],
      "supportedDecompressTypes": [
        "application/gzip",
        "application/x-bzip2",
        "application/zstd"
      ],
      "supportedDeltaTypes": [
        "application/x-rdiff-delta",
        "text/x-diff"
      ],
      "supportedPatchTypes": [
        "application/x-rdiff-delta",
        "text/x-diff"
      ],
      "maxConvertSize": 104857600,
      "maxArchiveEntries": 10000,
      "maxImageDimension": 8192
    }
  }
```

3. DataSourceObject

A DataSourceObject describes a source of data for use in Blob/set. It MUST contain exactly one of either data:asText or data:asBase64 (for inline data) or blobId (for a reference to an existing blob). The data:asText and data:asBase64 properties MUST NOT both be present in the same DataSourceObject.

* data:asText: "String|null"

  The data represented as a UTF-8 string. The server MUST reject the creation with a "notCreated" response if the value is not valid UTF-8.

* data:asBase64: "String|null"

  The data encoded as base64 ([BASE64]). The server MUST reject the creation with a "notCreated" response if the value is not valid base64.

* blobId: "BlobId"

  The blobId of an existing blob whose octets are to be copied into the new blob.

* offset: "UnsignedInt|null"

  The offset (in octets) within the data source from which to start copying. If null, defaults to 0 (the start of the data source). If the range cannot be fully satisfied (i.e., it begins or extends past the end of the data), the DataSourceObject is invalid and the creation MUST be rejected with a "notCreated" response.

* length: "UnsignedInt|null"

  The number of octets to copy from the data source. If null, copy from offset to the end of the data source. If the range cannot be fully satisfied, the DataSourceObject is invalid and the creation MUST be rejected with a "notCreated" response.

* size: "UnsignedInt|null"

  The full size of the data source in octets. In the chunks array of a Blob/get response this is the size of the underlying chunk blob.

* position: "UnsignedInt|null"

  The byte offset of the start of this data source within the outer (containing) blob. Only meaningful in the chunks array of a Blob/get response.

A DataSourceObject in the chunks array of a Blob/get response MAY also include digest:* properties (e.g. digest:sha-256) when those names are included in the dataSourceProperties request argument. Each digest value is computed over the octets that the chunk contributes to the blob (i.e. after applying offset and length).

When a server provides size, position, or digest:* values in a Blob/get response, it MUST calculate them correctly. When a DataSourceObject containing size, position, or digest:* values is used in Blob/set, the server MUST reject the object if any provided value does not match the actual data.

4. Blob/set

Blob/set is defined under the urn:ietf:params:jmap:blob2 capability and requires that capability in the request’s using array.

This method creates, touches, and destroys blobs. It is a standard Foo/set method ([JMAP-CORE], Section 5.3) with the following blob-specific behaviours:

* create: "Id[BlobCreateObject]|null"

  A map of a creation id to a BlobCreateObject describing the blob to create, or null if no blobs are to be created.

* update: A map of blobId to a patch object. The only property a client can request to change is expires; all other properties can only be set to their existing values. The server can honour the requested expires (by not returning an expires in the updated BlobObject) or return a different expires value if it applied a different lifetime. The purpose of update is to "touch" the blob, refreshing its lifetime on the server. If the blobId does not exist, it MUST be reported in a notUpdated map with a "notFound" SetError.

* destroy: The server MUST reject any blob that is still referenced by another object with a "blobHasReference" SetError.

An BlobCreateObject has the following properties:

* data: "DataSourceObject[]"

  An array of zero or more DataSourceObjects. The new blob is formed by concatenating the data described by each DataSourceObject in the order given.

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

  A media type hint ([MEDIA-TYPES]) for the blob. The server MAY use this when setting the Content-Type of a subsequent download of the blob, but is not required to. The server MAY infer the type itself and override the client’s hint.

* noPersist: "Boolean" (default: false)

  If true, the resulting blob is ephemeral: it may be referenced via creation id backreferences within the same JMAP request, but the server is not required to persist it beyond the lifetime of the request. The server MAY omit ephemeral blobs from the created map of the response and from the createdIds of the final Response object if it did not create a referenceable blob. This allows servers to optimise pipelines where intermediate blobs are never needed after the request completes.

The server MUST NOT guess the user's intent when a DataSourceObject is invalid (e.g., contains non-UTF-8 data in data:asText or invalid base64 in data:asBase64). It MUST reject the creation and return a "notCreated" response for that creation id.

The response is a standard Foo/set response. The objects in the created and updated maps are BlobObjects. If an update set a new expires, the server returns a BlobObject with the actual expires that was applied; if the server accepted the requested value it need not return an expires in the BlobObject.

A BlobObject has the following properties:

* id: "BlobId"

  The id of the blob.

* type: "String|null"

  The media type of the blob, or null if unknown.

* size: "UnsignedInt"

  The size of the blob in octets.

* expires: "UTCDate|null"

  See "The expires Response Property" below.

5. Blob/get

Blob/get is defined under the urn:ietf:params:jmap:blob2 capability and requires that capability in the request's using array.

This method retrieves blob data. It is a standard Foo/get method ([JMAP-CORE], Section 5.1) with the following additional parameters and a custom response.

In addition to the standard accountId, ids, and properties arguments, Blob/get accepts:

* offset: "UnsignedInt|null" (default: null)

  The starting byte position within the blob. If null, defaults to 0.

* length: "UnsignedInt|null" (default: null)

The number of bytes to return. If null, returns all bytes from offset to the end of the blob.

* dataSourceProperties: "String[]" (default: ["blobId", "size"])

  If supplied, only the properties listed in the array are returned for each DataSourceObject in the chunks array. Available properties include blobId, size, offset, length, position, and digest:* values where * is a hash algorithm name from the IANA "Hash Function Textual Names" registry (e.g. digest:sha-256).

The requestable properties in the properties array are:

* data:asText

  The data in the selected range as a UTF-8 string. If the selected bytes are not valid UTF-8 (including truncation in the middle of a multi-octet sequence), the response value is null and isEncodingProblem is set to true.

* data:asBase64

  The data in the selected range encoded as base64 ([BASE64]).

* data

  Returns data:asText if the selected bytes are valid UTF-8, otherwise returns data:asBase64.

* digest:<algorithm>

  The base64-encoded digest of the selected range, computed using the named algorithm. The algorithm name MUST appear in the server's supportedDigestAlgorithms capability.

* size

  The total size of the blob in octets, regardless of the range selected.

* imageData

If the blob is an image or video type and the server can extract metadata, this is an ImageData object. Otherwise null. Only returned if explicitly requested in properties. The server MAY need to read the blob content to compute this data, so clients SHOULD only request it when needed. Since blobs are immutable, the result for a given blobId will never change, and the server SHOULD cache it.

An ImageData object has the following properties:

- width: "UnsignedInt|null" Width of the image in pixels, if known.

- height: "UnsignedInt|null" Height of the image in pixels, if known.

- orientation: "UnsignedInt|null" EXIF orientation value (1-8) as defined in the EXIF specification (CIPA DC-008). If present, indicates how the image should be rotated/flipped for display.

- date: "UTCDate|null" Date the image or video was captured (from EXIF DateTimeOriginal or equivalent), if present.

- gps: "ImageGPS|null" GPS coordinates from EXIF, if present. An ImageGPS object has the following properties:

  o latitude: "Number" — Latitude in decimal degrees.

  o longitude: "Number" — Longitude in decimal degrees.

- duration: "Number|null" Duration in seconds for video content, if known. This is a floating-point number to allow sub-second precision.

- comment: "String|null" Embedded EXIF comment or description, if present. This is read-only — it reflects the metadata stored in the blob content and is not a user-editable property.

If neither offset nor length are specified (i.e., both null), the default properties are data and size. Otherwise, the client MUST specify the properties to return.

The response contains a list of objects, one per blobId. Each object contains the requested properties plus:

* id: "BlobId"

  The blobId of the blob.

* isEncodingProblem: "Boolean" (default: false)

  Set to true if data or data:asText was requested but the selected range is not valid UTF-8.

* isTruncated: "Boolean" (default: false)

  Set to true if the selected range extends past the end of the blob. In this case, only the bytes from offset to the end of the blob (or an empty string if offset is past the end) are returned.

* chunks: "DataSourceObject[]"

  An array of one or more data source objects describing the internal chunk structure of the blob. The blob is reconstructed by concatenating the data from each data source object in the listed order. The client MUST use the offset and length of each DataSourceObject to determine which octets to read from each chunk's underlying blob.

  This property MUST be explicitly requested in properties.

The response also contains:

* accountId: "Id"

  The id of the account used for the operation.

* notFound: "Id[]"

  An array of blobIds that were not found.

6. Blob/lookup

Blob/lookup is defined under the urn:ietf:params:jmap:blob2 capability and requires that capability in the request's using array.

This method provides a reverse lookup: given a set of blobIds, it returns which objects of specified data types reference those blobs.

The method takes the following arguments:

* accountId: "Id"

  The id of the account to use.

* typeNames: "String[]"

A list of data type names to search. Only type names listed in the server's supportedTypeNames capability are valid. If a type name is not known by the server or the associated capability has not been requested in using, the server MUST return an "unknownDataType" error.

* ids: "BlobId[]"

  The blobIds to look up.

The response contains:

* accountId: "Id"

  The id of the account used for the operation.

* list: "BlobInfo[]"

  A list of BlobInfo objects, one per blobId. A BlobInfo object has the following properties:

  - id: "BlobId" The blobId queried.

  - matchedIds: "String[Id[]]" An object mapping each requested type name to an array of object ids of that type that reference this blob. If no objects of that type reference this blob, the array is empty.

Access control: if a blob is not visible to the authenticated user or does not exist on the server at all, the server MUST still return an empty array for each type name. This ensures that the response does not reveal whether the blob exists but is inaccessible to the user.

7. The "expires" Response Property

The Blob/set response and the Blob/convert response include the following property on each BlobObject:

* expires: "UTCDate|null"

  A hint from the server indicating the likely availability of the blob. The blob is likely to remain available until this time, and likely not to be available after it. This is not a guarantee in either direction: the server MAY garbage collect the blob before this time if it is unreferenced, and MAY retain it longer. If null, the server does not have a specific expiry time for the blob.

Clients that need the blob to persist beyond the expires time should reference it from a persistent object (e.g., a FileNode or an Email) before it expires.

8. Blob/convert

Blob/convert is defined under the urn:ietf:params:jmap:blob2 capability and requires that capability in the request's using array.

Blob/convert performs server-side transformations on blobs. Like Blob/set, it takes an accountId and a create argument that maps creation ids to conversion request objects.

Each conversion request object MAY also include the following property:

* noPersist: "Boolean" (default: false) If true, the resulting blob is ephemeral: it may be referenced via creation id backreferences within the same JMAP request, but the server is not required to persist it beyond the lifetime of the request. The server MAY omit ephemeral blobs from the created map of the response and from the createdIds of the final Response object if it did not create a referenceable blob.

Each conversion request object MUST contain exactly one of the following properties, which determines the type of conversion:

* imageConvert: ImageConvertRecipe

* archive: ArchiveRecipe

* extract: ExtractRecipe

* compress: CompressRecipe

* decompress: DecompressRecipe

* delta: DeltaRecipe

* patch: PatchRecipe

The response has the same structure as Blob/set: a created map of creation id to a BlobObject for each successful conversion, and a notCreated map of creation id to a SetError object for each failed conversion. The id is the blobId of the created blob. The response also includes an expires property on each BlobObject (see "The expires Response Property" above).

If the conversion completed but encountered problems (e.g., a corrupt archive where some entries extracted successfully but others did not, or a truncated compressed stream that partially decompressed), the response for that creation also includes:

* isIncomplete: "Boolean" (default: false) If true, the conversion did not complete cleanly. The output blob may be partial or missing some content.

* description: "String|null" A human-readable description of the problem, if any.

If the conversion failed entirely and no usable output could be produced, the server MUST return a conversionFailed SetError in the notCreated map.

Creation id backreferences (using the # prefix) resolve to the id of the created blob and may be used in subsequent conversions within the same Blob/convert call or in later method calls within the same JMAP request.

A server MAY return a blobId for a conversion result without immediately generating the output data. In this case the server MUST generate the data when the blob is later accessed (e.g., via a download request or as input to another operation). This allows the server to respond quickly to Blob/convert requests while deferring expensive work such as image resizing or archive creation. The returned blobId MUST be usable in all contexts where a regular blobId is accepted. If the deferred generation later fails (e.g., the source blob has expired), the server SHOULD return an appropriate HTTP error when the blob is downloaded. If the exact size of the output is not yet known, the server MUST omit the size property from the response for that creation. If a client later requests the size property via Blob/get for a deferred blob, the server MUST generate the blob at that point and return the actual size.

The server MUST resolve the order of dependencies between entries in the create map and process them in an order such that all backreferences are satisfied. If a dependency cycle is detected, all members of the cycle MUST be rejected with an "invalidProperties" error.

8.1. ImageConvertRecipe

An ImageConvertRecipe converts an image blob to a different format or size. It is an object with the following properties:

* blobId: "BlobId" The blobId of the source image.

* type: "String" Media type ([MEDIA-TYPES]) of the image to create (e.g. "image/png"). MUST be one of the values in the server's supportedImageWriteTypes capability. The source image MUST be in a format listed in supportedImageReadTypes.

* width: "UnsignedInt|null" Maximum width in pixels of the image to create. If null, the server preserves the source width (or scales proportionally if only height is given).

* height: "UnsignedInt|null" Maximum height in pixels of the image to create. If null, the server preserves the source height (or scales proportionally if only width is given).

* ignoreAspect: "Boolean|null" If true, resize to exactly the given width and height, even if the aspect ratio is changed. If null or false, the image is scaled to fit within the given dimensions while preserving the aspect ratio.

* quality: "UnsignedInt|null" Compression quality for lossy formats, as a value from 1 (lowest quality, smallest file) to 100 (highest quality, largest file). Only meaningful for formats that support lossy compression such as image/jpeg and image/webp. If null, the server selects a sensible default.

* colorSpace: "String|null" The color space for the output image. Defined values are "sRGB" and "grayscale". If null, the server preserves the source image's color space where possible.

* background: "String|null" A fill color to use when the source image has transparency but the target format does not support it (e.g. converting PNG to JPEG). The value is a CSS-style hex color string (e.g. "#ffffff" for white). If null, the server selects a sensible default (typically white).

* stripMetadata: "Boolean|null" If true, strip image metadata such as EXIF, XMP, and IPTC data from the output. If null or false, the server preserves metadata where the target format supports it.

* autoOrient: "Boolean|null" If true, automatically rotate and flip the image according to its EXIF orientation tag, then reset the tag. If null or false, the image data is not reoriented.

Errors:

* "notFound" — the referenced blobId does not exist.

* "invalidProperties" — the type is not in supportedImageWriteTypes, or the source blob is not in a format listed in supportedImageReadTypes.

* "tooLarge" — the source blob exceeds maxConvertSize, or the requested dimensions exceed maxImageDimension.

* "conversionFailed" — the image conversion failed entirely.

8.2. ArchiveRecipe

An ArchiveRecipe creates an archive blob from a list of entries. It is an object with the following properties:

* type: "String" The media type ([MEDIA-TYPES]) of the archive to create. MUST be one of the values in the server's supportedArchiveTypes capability.

* entries: "ArchiveEntry[]" An array of ArchiveEntry objects describing the contents of the archive.

Errors:

* "notFound" — a referenced entry blobId does not exist.

* "invalidProperties" — the type is not in supportedArchiveTypes; an entry has an unsupported entryType for the archive format; or a required field (e.g. linkTarget for symlink entries) is missing.

* "tooLarge" — the number of entries exceeds maxArchiveEntries, or a referenced blob exceeds maxConvertSize.

* "conversionFailed" — the archive creation failed entirely.

8.3. ArchiveEntry

An ArchiveEntry describes a single entry in an archive. It is used both as input (in ArchiveRecipe) and as output (in ExtractRecipe results). It is an object with the following properties:

* name: "String" The path of the entry within the archive. Directory entries MUST have a name ending with "/".

* blobId: "BlobId|null" The blobId of the content for this entry. MUST be non-null for file entries. MUST be null or absent for directory, symlink, hardlink, fifo, and device entries. Violating these constraints is an "invalidProperties" error.

* entryType: "String|null" The type of the entry. If null, defaults to "file". Defined values are "file" (a regular file, the default), "directory", "symlink" (a symbolic link), "hardlink" (a hard link), "fifo" (a named pipe), "blockDevice" (a block device node), and "charDevice" (a character device node). The server MUST reject entries with unsupported types for the chosen archive format with an "invalidProperties" error.

* modified: "UTCDate|null" The modification time of the entry as an RFC 3339 timestamp. If null, defaults to the current server time.

* linkTarget: "String|null" The target path for symlink and hardlink entries. MUST be non-null when entryType is "symlink" or "hardlink". MUST be null for all other entry types.

* mode: "String|null" Unix file permissions as an octal string (e.g. "0755", "0644"). If null, the server chooses a reasonable default. This is represented as a string rather than an integer to avoid ambiguity between octal and decimal interpretation.

* uid: "UnsignedInt|null" The numeric user ID of the entry owner.

* gid: "UnsignedInt|null" The numeric group ID of the entry owner.

* ownerName: "String|null" The user name of the entry owner.

* groupName: "String|null" The group name of the entry owner.

* devMajor: "UnsignedInt|null" The major device number for "blockDevice" and "charDevice" entries.

* devMinor: "UnsignedInt|null" The minor device number for "blockDevice" and "charDevice" entries.

* comment: "String|null" A comment string for this entry.

* compressionMethod: "String|null" The per-entry compression method. Defined values are "store" (no compression) and "deflate". If null, the server chooses a reasonable default.

8.3.1. Considerations for application/zip

The zip format only supports "file" and "directory" entry types. The comment and compressionMethod properties are only meaningful for zip archives. The mode, uid, gid, ownerName, groupName, devMajor, and devMinor properties are ignored.

8.3.2. Considerations for application/x-tar

The tar format supports all entry types. The mode, uid, gid, ownerName, groupName, devMajor, and devMinor properties are meaningful for tar archives. The comment and compressionMethod properties are ignored.

Tar archives are not inherently compressed. To create a compressed tar archive (e.g. a .tar.gz file), first create the tar archive using ArchiveRecipe, then compress the result using CompressRecipe. The following example creates a .tar.gz containing three files in a single Blob/convert call, using a backreference from the archive creation to the compression step:

```
[["Blob/convert", {
  "accountId": "abc",
  "create": {
    "t1": {
      "archive": {
        "type": "application/x-tar",
        "entries": [
          {
            "name": "site/index.html",
            "blobId": "Baaaa",
            "modified": "2026-03-01T12:00:00Z",
            "mode": "0644"
          },
          {
            "name": "site/logo.png",
            "blobId": "Bbbbb",
            "modified": "2026-02-15T09:30:00Z",
            "mode": "0644"
          },
          {
            "name": "site/style.css",
            "blobId": "Bcccc",
            "modified": "2026-03-01T12:00:00Z",
            "mode": "0644"
          }
        ]
      }
    },
    "t2": {
      "compress": {
        "blobId": "#t1",
        "type": "application/gzip"
      }
    }
  }
}, "0"]]
```

8.3.3. Considerations for application/x-cpio

The cpio format supports all entry types except that ownerName and groupName are not supported. The comment and compressionMethod properties are ignored.

8.4. ExtractRecipe

An ExtractRecipe extracts the entry listing from an existing archive blob. It is an object with the following properties:

* blobId: "BlobId" The blobId of the archive to extract.

* type: "String|null" The MIME type of the archive format. If null, the server SHOULD attempt to auto-detect the format from the blob content (e.g. by inspecting magic bytes). If auto-detection fails, the server MUST return an "unknownFormat" error.

In addition to the standard creation response properties, a successful ExtractRecipe result includes:

* entries: "ArchiveEntry[]" An array of ArchiveEntry objects describing the contents of the archive. Each file entry will have a blobId that can be used to access the content of that entry.

Errors:

* "notFound" — the referenced blobId does not exist.

* "unknownFormat" — the server could not determine or does not support the archive format.

* "invalidProperties" — the type is not in supportedExtractTypes.

* "tooLarge" — the source blob exceeds maxConvertSize.

* "conversionFailed" — the extraction failed entirely.

8.5. CompressRecipe

A CompressRecipe compresses a blob using a specified compression algorithm. It is an object with the following properties:

* blobId: "BlobId" The blobId of the data to compress.

* type: "String" The media type ([MEDIA-TYPES]) of the compression format to use. MUST be one of the values in the server's supportedCompressTypes capability.

* level: "UnsignedInt|null" The compression level, where higher values produce smaller output at the cost of more CPU time. If null, the server uses the format's default level. The valid range depends on the format; if the requested level is outside the valid range, the server SHOULD use the nearest valid value.

* checksum: "Boolean|null" If true, include an integrity checksum in the compressed output. If null, the server uses the format's default behaviour.

Errors:

* "notFound" — the referenced blobId does not exist.

* "invalidProperties" — the type is not in supportedCompressTypes.

* "tooLarge" — the source blob exceeds maxConvertSize.

* "conversionFailed" — the compression failed entirely.

8.5.1. Considerations for application/gzip

Compression level ranges from 1 (fastest) to 9 (best compression). The default is typically 6. Gzip always includes a CRC-32 checksum; the checksum property is ignored.

8.5.2. Considerations for application/x-bzip2

Compression level ranges from 1 (fastest, 100k block size) to 9 (best compression, 900k block size). The default is typically 9. Bzip2 always includes a CRC-32 checksum; the checksum property is ignored.

8.5.3. Considerations for application/x-xz

Compression level ranges from 0 (fastest) to 9 (best compression). The default is typically 6. Xz always includes an integrity check; if checksum is true the server SHOULD use SHA-256, otherwise CRC-64 is used by default.

8.5.4. Considerations for application/zstd

Compression level ranges from 1 (fastest) to 22 (best compression). The default is typically 3. If checksum is true, an xxHash-64 checksum is included in the frame; the default is false.

8.6. DecompressRecipe

A DecompressRecipe decompresses a compressed blob. It is an object with the following properties:

* blobId: "BlobId" The blobId of the compressed data to decompress.

* type: "String|null" The MIME type of the compression format. If null, the server SHOULD attempt to auto-detect the format from magic bytes. If auto-detection fails, the server MUST return an "unknownFormat" error.

Errors:

* "notFound" — the referenced blobId does not exist.

* "unknownFormat" — the server could not determine or does not support the compression format.

* "invalidProperties" — the type is not in supportedDecompressTypes.

* "tooLarge" — the source blob exceeds maxConvertSize.

* "conversionFailed" — the decompression failed entirely.

8.7. DeltaRecipe

A DeltaRecipe computes a delta between two blobs. It is an object with the following properties:

* blobId: "BlobId" The blobId of the original (base) blob.

* newBlobId: "BlobId" The blobId of the new blob to compare against.

* type: "String" The media type of the delta format to produce. MUST be one of the values in the server’s supportedDeltaTypes capability.

The result blob is the computed delta, which can be applied to the base blob using PatchRecipe to reconstruct the new blob.

Errors:

* "notFound" — a referenced blobId does not exist.

* "invalidProperties" — the type is not in supportedDeltaTypes.

* "tooLarge" — a referenced blob exceeds maxConvertSize.

* "conversionFailed" — the delta computation failed entirely.

8.7.1. Considerations for application/x-rdiff-delta

The server computes an rdiff signature of the base blob and then generates a delta against the new blob. The resulting delta blob can only be applied to the exact base blob used to generate it.

8.7.2. Considerations for application/x-bsdiff

The server produces a bsdiff-format patch. Both blobs must fit in memory; servers MAY reject very large blobs with a "tooLarge" error.

8.7.3. Considerations for text/x-diff

The server produces a unified diff. Both blobs are interpreted as text. If either blob contains content that cannot be interpreted as text, the server MUST return an "unknownFormat" error.

8.8. PatchRecipe

A PatchRecipe applies a delta to a base blob to produce a new blob. It is an object with the following properties:

* blobId: "BlobId" The blobId of the base blob to patch.

* deltaBlobId: "BlobId" The blobId of the delta to apply.

* deltaType: "String" The media type of the delta format. MUST be one of the values in the server's supportedPatchTypes capability.

The result blob is the patched output.

Errors:

* "notFound" — a referenced blobId does not exist.

* "unknownFormat" — the delta blob is not valid for the specified format (e.g., corrupt or malformed delta data).

* "invalidProperties" — the deltaType is not in supportedPatchTypes.

* "tooLarge" — a referenced blob exceeds maxConvertSize.

* "conversionFailed" — the patch application failed entirely.

9. Examples

9.1. Uploading a Blob Inline

This example creates a blob from inline text data:

```
[["Blob/set", {
  "accountId": "abc",
  "create": {
    "b1": {
      "data": [{"data:asText": "Hello, world!"}],
      "type": "text/plain"
    }
  }
}, "0"]]
```

The response:

```
[["Blob/set", {
  "accountId": "abc",
  "created": {
    "b1": {
      "id": "Babc123",
      "type": "text/plain",
      "size": 13
    }
  },
  "notCreated": null
}, "0"]]
```

9.2. Querying Blob Chunks

This example fetches a blob’s chunk structure with offsets, sizes, and SHA-256 digests:

```
[["Blob/get", {
  "accountId": "abc",
  "ids": ["B1a2b3c"],
  "dataSourceProperties": [
    "blobId", "size", "offset", "length", "position",
    "digest:sha-256"
  ]
}, "0"]]
```

The response shows the blob is stored as two chunks:

```
[["Blob/get", {
  "accountId": "abc",
  "list": [{
    "id": "B1a2b3c",
    "size": 10485760,
    "chunks": [
      {
        "blobId": "Bchunk1",
        "size": 5242880,
        "offset": 0,
        "length": 5242880,
        "position": 0,
        "digest:sha-256": "a1b2c3..."
      },
      {
        "blobId": "Bchunk2",
        "size": 5242880,
        "offset": 0,
        "length": 5242880,
        "position": 5242880,
        "digest:sha-256": "d4e5f6..."
      }
    ]
  }],
  "notFound": []
}, "0"]]
```

The position values show where each chunk fits in the assembled blob, and the digest:sha-256 values can be used to verify chunk integrity.

9.3. Creating a Zip Archive

This example creates a zip file containing an HTML document, a CSS file, and a JPEG image:

```
[["Blob/convert", {
  "accountId": "abc",
  "create": {
    "z1": {
      "archive": {
        "type": "application/zip",
        "entries": [
          {
            "name": "site/index.html",
            "blobId": "Baaaa"
          },
          {
            "name": "site/style.css",
            "blobId": "Bbbbb"
          },
          {
            "name": "site/photo.jpg",
            "blobId": "Bcccc"
          }
        ]
      }
    }
  }
}, "0"]]
```

The response includes the blobId, type, and size of the created archive:

```
[["Blob/convert", {
  "accountId": "abc",
  "created": {
    "z1": {
      "id": "B9f2a4e",
      "type": "application/zip",
      "size": 104857
    }
  },
  "notCreated": {}
}, "0"]]
```

9.4. Creating a Compressed Tar Archive

This example creates a .tar.gz file from the same three files. The intermediate tar blob uses noPersist since only the final compressed result is needed:

```
[["Blob/convert", {
  "accountId": "abc",
  "create": {
    "t1": {
      "noPersist": true,
      "archive": {
        "type": "application/x-tar",
        "entries": [
          {
            "name": "site/index.html",
            "blobId": "Baaaa",
            "modified": "2026-03-01T12:00:00Z",
            "mode": "0644"
          },
          {
            "name": "site/style.css",
            "blobId": "Bbbbb",
            "modified": "2026-03-01T12:00:00Z",
            "mode": "0644"
          },
          {
            "name": "site/photo.jpg",
            "blobId": "Bcccc",
            "modified": "2026-02-15T09:30:00Z",
            "mode": "0644"
          }
        ]
      }
    },
    "t2": {
      "compress": {
        "blobId": "#t1",
        "type": "application/gzip"
      }
    }
  }
}, "0"]]
```

Because "t1" was created with noPersist, the server may omit it from the response:

```
[["Blob/convert", {
  "accountId": "abc",
  "created": {
    "t2": {
      "id": "Bd81c7f",
      "type": "application/gzip",
      "size": 98304
    }
  },
  "notCreated": {}
}, "0"]]
```

9.5. Extracting a Compressed Tar Archive

This example decompresses and extracts a .tar.gz file to discover its contents. The intermediate decompressed tar blob uses noPersist:

```
[["Blob/convert", {
  "accountId": "abc",
  "create": {
    "u1": {
      "noPersist": true,
      "decompress": {
        "blobId": "Bd81c7f",
        "type": "application/gzip"
      }
    },
    "u2": {
      "extract": {
        "blobId": "#u1",
        "type": "application/x-tar"
      }
    }
  }
}, "0"]]
```

The response for the ExtractRecipe includes the standard creation response properties plus an entries array listing the archive contents, with a blobId for each file entry:

```
[["Blob/convert", {
  "accountId": "abc",
  "created": {
    "u2": {
      "id": "Be3a901",
      "type": "application/x-tar",
      "size": 102400,
      "entries": [
        {
          "name": "site/index.html",
          "blobId": "Bdd001",
          "entryType": "file",
          "modified": "2026-03-01T12:00:00Z",
          "mode": "0644"
        },
        {
          "name": "site/style.css",
          "blobId": "Bdd002",
          "entryType": "file",
          "modified": "2026-03-01T12:00:00Z",
          "mode": "0644"
        },
        {
          "name": "site/photo.jpg",
          "blobId": "Bdd003",
          "entryType": "file",
          "modified": "2026-02-15T09:30:00Z",
          "mode": "0644"
        }
      ]
    }
  },
  "notCreated": {}
}, "0"]]
```

The returned blobIds ("Bdd001", "Bdd002", "Bdd003") can be used to download individual files or as inputs to further Blob/convert operations.

9.6. Computing and Applying a Delta

This example computes a unified diff between two versions of a text file:

```
[["Blob/convert", {
  "accountId": "abc",
  "create": {
    "d1": {
      "delta": {
        "blobId": "BoldVersion",
        "newBlobId": "BnewVersion",
        "type": "text/x-diff"
      }
    }
  }
}, "0"]]
```

The response contains the delta blob:

```
[["Blob/convert", {
  "accountId": "abc",
  "created": {
    "d1": {
      "id": "Bdelta789",
      "type": "text/x-diff",
      "size": 1234
    }
  },
  "notCreated": {}
}, "0"]]
```

The delta can later be applied to the original blob to reconstruct the new version:

```
[["Blob/convert", {
  "accountId": "abc",
  "create": {
    "p1": {
      "patch": {
        "blobId": "BoldVersion",
        "deltaBlobId": "Bdelta789",
        "deltaType": "text/x-diff"
      }
    }
  }
}, "0"]]
```

The result is a blob identical to "BnewVersion":

```
[["Blob/convert", {
  "accountId": "abc",
  "created": {
    "p1": {
      "id": "Breconstructed",
      "type": "application/octet-stream",
      "size": 48576
    }
  },
  "notCreated": {}
}, "0"]]
```

10. Security Considerations

All security considerations from [JMAP-CORE] apply to this document.

10.1. Access Control

Servers MUST NOT allow a client to access blob data that the authenticated user does not have permission to access. When a DataSourceObject references a blobId, the server MUST verify that the requesting user has access to that blob.

10.2. Untrusted Data

Clients that create blobs from inline data (data:asText or data:asBase64) MUST NOT assume that the server will accept all possible values. Servers MUST reject malformed inputs as specified in the DataSourceObject definition.

10.3. Resource Consumption

Several operations defined in this document can consume significant server resources.

Archive and compression operations may require substantial CPU and memory. Servers SHOULD impose reasonable limits on archive size, number of entries, nesting depth of archives within archives, compression ratios (to mitigate zip bomb attacks), and total processing time. Servers SHOULD reject requests that would exceed these limits with a "tooLarge" or "serverFail" error as appropriate.

Image conversion can also consume significant resources, especially for very large images or high output dimensions.

Delta and patch operations can also be resource-intensive, particularly for large blobs. Servers SHOULD impose limits on the size of blobs that can be used as inputs to these operations.

Servers SHOULD advertise their limits via the maxConvertSize, maxArchiveEntries, and maxImageDimension capability properties so that clients can avoid making requests that will be rejected. Even when these properties are not advertised, servers SHOULD set sensible internal limits and reject requests that exceed them.

10.4. Archive Path Traversal

Archive formats allow entry names containing path separators and relative path components such as "../". Malicious archives may use names like "../../etc/passwd" to attempt directory traversal. While ExtractRecipe only returns entry metadata and blob references (not files on the server filesystem), clients that extract archive contents to a filesystem MUST validate entry names and reject or sanitize paths containing ".." components or absolute paths. Servers SHOULD reject ArchiveRecipe requests containing entry names with ".." path components.

10.5. Content Smuggling

The ability to split content into multiple blobs, recombine them via Blob/set, and apply delta patches may be used to bypass security scanners that inspect blob content. Servers that perform content scanning SHOULD scan the output of Blob/convert operations as well as the inputs.

11. IANA Considerations

11.1. JMAP Capability Registration for urn:ietf:params:jmap:blob2

IANA is requested to register the "Blob Management" Capability as follows:

Capability Name: urn:ietf:params:jmap:blob2

Intended use: common

Change Controller: IETF

Specification document: this document

Security and privacy considerations: this document, Security Considerations

11.2. JMAP Error Code Registrations

IANA is requested to register the following entries in the "JMAP Error Codes" registry:

11.2.1. unknownDataType

JMAP Error Code: unknownDataType

Intended use: common

Change Controller: IETF

Description: The server does not recognise the data type specified in the typeNames array of a Blob/lookup request, or the capability required to use that data type has not been included in the using array of the request.

Reference: this document

11.2.2. unknownFormat

JMAP Error Code: unknownFormat

Intended use: common

Change Controller: IETF

Description: The server could not determine the format of the blob, or the detected format is not supported. This error is returned when auto-detection of archive or compression format fails, or when the blob content does not match the specified format.

Reference: this document

11.2.3. blobHasReference

JMAP Error Code: blobHasReference

Intended use: common

Change Controller: IETF

Description: The blob cannot be destroyed because it is still referenced by one or more objects. The client can use Blob/lookup to discover which objects reference the blob.

Reference: this document

11.2.4. conversionFailed

JMAP Error Code: conversionFailed

Intended use: common

Change Controller: IETF

Description: The server was unable to perform the requested blob conversion.

Reference: this document

11.3. JMAP Data Types Registry

IANA is requested to update the reference for the "JMAP Data Types" registry to this document (replacing the reference to RFC9404).

12. Changes

EDITOR: please remove this section before publication.

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

draft-ietf-jmap-blobext-01

* Added per-account uploadUrl capability property, allowing servers to direct uploads for individual accounts to a different endpoint. Uploads to this URL return an expires property in addition to the standard upload response fields.

* Blob/set is now a standard Foo/set method with state strings (ifInState/oldState/newState). Renamed UploadObject to BlobCreateObject. The expires property can be set via update.

* Removed hardcoded type value lists from normative text (supportedArchiveTypes, supportedCompressTypes, supportedDeltaTypes, ArchiveRecipe, CompressRecipe). Values are now described as media types per [MEDIA-TYPES]. Format-specific considerations subsections remain as informative examples.

* Obsoletes RFC9404: incorporated all content from RFC9404 into this document, making it self-contained.

* Renamed Blob/upload to Blob/set, making it a standard Foo/set operation with create, update, and destroy at the top level.

* Added urn:ietf:params:jmap:blob capability definition (previously defined only in RFC9404).

* Added Blob/lookup method (previously defined only in RFC9404).

* Added full DataSourceObject definition (merging RFC9404 base definition with blobext extensions).

* Added full Blob/get definition (merging RFC9404 base definition with blobext extensions).

* Added inline blob creation example.

* Updated IANA section to include blob capability, unknownDataType error code, and JMAP Data Types registry.

draft-ietf-jmap-blobext-00

* No changes, just uploading with the new name

draft-gondwana-jmap-blobext-06

* Removed resumableUploadUrl capability property.

* Added isIncomplete and description response properties for partial conversion results.

* Added conversionFailed error code for total conversion failure.

* Made expires always present (null if indeterminate) rather than optional.

* Split supportedImageTypes into supportedImageReadTypes and supportedImageWriteTypes.

* Made all supported type lists nullable (null = not supported).

* Changed imageData duration from UnsignedInt to Number for sub-second precision.

* Clarified imageData comment as read-only.

* Added EXIF orientation reference and digest algorithm registry reference.

* Fixed grammar and consistent error ordering across recipes.

* Structural reorder: DataSourceObject before Blob/get, ArchiveEntry promoted to peer section.

draft-gondwana-jmap-blobext-05

* Added imageData property to Blob/get for image/video metadata extraction (dimensions, orientation, date, GPS, duration).

* Renamed UnArchiveRecipe to ExtractRecipe and UnCompressRecipe to DecompressRecipe.

* Split capability lists into separate create/extract pairs: supportedArchiveTypes/supportedExtractTypes, supportedCompressTypes/supportedDecompressTypes, supportedDeltaTypes/supportedPatchTypes.

draft-gondwana-jmap-blobext-04

* Moved noPersist from top-level Blob/upload property to per-item in the create map, for consistency with Blob/convert.

draft-gondwana-jmap-blobext-03

* Added update (touch) and destroy operations to Blob/upload.

draft-gondwana-jmap-blobext-02

* Replaced RdiffRecipe with generic DeltaRecipe and PatchRecipe using media types (rdiff, bsdiff, unified diff).

* Replaced supportsRdiff with supportedPatchTypes capability.

* Clarified chunkSize as a hint, not a definitive statement.

* Added lazy generation text for Blob/convert (deferred output).

* Defined "expires" response property for blob creation responses.

draft-gondwana-jmap-blobext-01

* Added Blob/convert method with recipes for image conversion, archiving, compression, and rdiff.

* Added noPersist option to Blob/upload and Blob/convert for ephemeral intermediate blobs in pipelines.

* Removed rdiffSignature and rdiffPatch from Blob/get and DataSourceObject.

* Fleshed out capability object with supported type lists.

* Added capability and method examples.

* Expanded Security Considerations.

* Updated IANA registrations with error codes and corrected capability URI.

* Added limit capability properties: maxConvertSize, maxArchiveEntries, maxImageDimension, and supportsRdiff.

* Added dependency resolution requirement for Blob/convert create map with cycle detection.

* Now updates both RFC 8620 and RFC 9404.

draft-gondwana-jmap-blobext-00

* initial proposal

13. Acknowledgements

The authors would like to thank Mauro De Gennaro, Ben Bucksch, Ricardo Signes, and the members of the JMAP Working Group at the IETF for their contributions and feedback.

{backmatter}

14. Normative References

[BASE64] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[DIGEST-ALGORITHMS] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/rfc/rfc3230>.

[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>.

[MEDIA-TYPES] 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>.

[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>.

[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>.

Author’s Address

Bron Gondwana
Fastmail
Email: brong@fastmailteam.com