Network Working Group
Internet-Draft
Updates: 8620, 9404 (if approved)
Intended status: Standards Track
Expires: 19 September 2026

B. Gondwana
Fastmail

18 March 2026

# JMAP Blob Extensions
## draft-gondwana-jmap-blobext-03

## 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.

The JMAP blob extension (RFC9404) added additional ways to create and access blobs by making inline method calls within a standard JMAP request.

This extension adds more methods to work with blobs, including handling large blobs by processing them in chunks (building on RFC9404's blob construction support), and providing server-side blob conversion operations: image format conversion, archive creation and extraction (zip, tar, cpio), compression and decompression, and delta/patch operations.

## 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 19 September 2026.

## Copyright Notice

Table of Contents

# 1. Introduction

The JMAP Blob extension ([JMAP-BLOB] — JMAP Blob Management) offers additional ways to create blobs, and query where they are used.

This extension builds on that work, offering ways to find more information about the internal structure of the server's blob store in order to work efficiently with it, and a new Blob/convert method for server-side blob transformations including image format conversion, archive creation and extraction, compression and decompression, and delta/patch operations.

## 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 an additional capability URI.

## 2.1. urn:ietf:params:jmap:blobext

The capability urn:ietf:params:jmap:blobext being present in the "accountCapabilities" property of an account represents support for these extended properties on that account. This capability depends on urn:ietf:params:jmap:blob ([JMAP-BLOB]); both MUST be present in the account's "accountCapabilities" and in the request's using array. If this capability is present in one or more "accountCapabilities" properties then the server MUST also include the key 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:

* resumableUploadUrl: "String|null"

If present, a [URI-TEMPLATE] which supports [HTTP-RESUMABLE-UPLOADS]. This MAY be the same as the uploadUrl, and has the same keys.

* 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/upload 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/upload though, and will then choose whether to store them as an array of blobs still, or to combine them.

* supportedImageTypes: "String[]"

  The media types ([MEDIA-TYPES]) supported for ImageConvertRecipe.

* supportedArchiveTypes: "String[]"

  The archive MIME types supported for ArchiveRecipe and UnArchiveRecipe. Defined values are "application/zip", "application/x-tar", and "application/x-cpio".

* supportedCompressionTypes: "String[]"

  The compression MIME types supported for CompressRecipe and UnCompressRecipe. Defined values are "application/gzip", "application/x-bzip2", "application/x-xz", and "application/zstd".

* supportedPatchTypes: "String[]"

  The delta/patch media types supported for DeltaRecipe and PatchRecipe. Defined values are "application/x-rdiff-delta", "application/x-bsdiff", and "text/x-diff" (unified diff). An empty array means no delta/patch operations are supported.

* 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:blobext": {
    "resumableUploadUrl": null,
    "chunkSize": 5242880,
    "supportedImageTypes": [
      "image/png",
      "image/jpeg",
      "image/gif"
    ],
    "supportedArchiveTypes": [
      "application/zip",
      "application/x-tar"
    ],
    "supportedCompressionTypes": [
      "application/gzip",
      "application/x-bzip2",
      "application/zstd"
    ],
    "supportedPatchTypes": [
      "application/x-rdiff-delta",
      "text/x-diff"
    ],
    "maxConvertSize": 104857600,
    "maxArchiveEntries": 10000,
    "maxImageDimension": 8192
  }
}
```

### 2.2. Additions to Blob/get

When this capability is present, Blob/get accepts an additional request argument:

* dataSourceProperties: "String[]" (default: ["blobId", "size"]) If supplied, only the properties listed in the array are returned for each DataSourceObject in the chunks array. If omitted, the default of ["blobId", "size"] is used. Available properties include blobId, size, offset, length, position, and digest:* values (e.g. digest:sha-256).

Blob/get also returns an additional response property (returned by default when this capability is included in using):

* chunks: "DataSourceObject[]"

  An array of one or more data source objects (as defined in [JMAP-BLOB], Section 4.2). The blob is reconstructed by concatenating the data from each data source object in the listed order. While it is expected that each data source object will reference the entire underlying chunk blob, the server MAY return offset and length values that select only a portion of a chunk's blob. The client MUST use the offset and length to determine which octets to read from each chunk.

## 2.3. Additions to DataSourceObject

When this capability is present, the DataSourceObject (as defined in [JMAP-BLOB], Section 4.2) is extended with the following additional properties. These apply both to the chunks returned by Blob/get and to DataSourceObjects used in Blob/upload. The offset and length properties are already defined in [JMAP-BLOB]; they are listed here to document their use in the chunks response context, where they describe the range of each chunk's underlying data source that contributes to the containing blob.

* offset: "UnsignedInt|null" The offset within the data source from which to start copying octets (see [JMAP-BLOB]). MUST fit within the data source (i.e. offset MUST be less than or equal to the data source size). If null, defaults to 0 (the start of the data source).

* length: "UnsignedInt|null" The number of octets to copy from the data source (see [JMAP-BLOB]). MUST fit within the data source (i.e. offset + length MUST be less than or equal to the data source size). If null, copy from offset to the end of the data source.

* size: "UnsignedInt|null" The full size of the chunk's underlying data source in octets.

* position: "UnsignedInt|null" The byte offset of the start of this chunk within the outer (containing) blob.

If a digest:* property (e.g. digest:sha, digest:sha-256) is included in dataSourceProperties, each DataSourceObject in the chunks array will include the corresponding digest value computed over the octets that this chunk contributes (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/upload, the server MUST reject the object if any provided value does not match the actual data.

## 2.4. Additions to Blob/upload

When this capability is present, Blob/upload ([JMAP-BLOB], Section 4) gains the following additional request property:

* noPersist: "Boolean" (default: false) If true, blobs created by this call are ephemeral: they may be referenced via creation id backreferences within the same JMAP request, but the server is not required to persist them 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.

When this capability is present, Blob/upload also gains the following additional request properties:

* update: "Id[Object]" (default: null) A map of blobId to an empty object. No properties may be set on a blob; the purpose of update is to "touch" the blob, refreshing its lifetime on the server. The response includes an updated map of blobId to an object which MAY contain an expires property with the new expiry time. If the blobId does not exist, it MUST be reported in a notUpdated map with a notFound SetError.

* destroy: "Id[]" (default: null) An array of blobIds to destroy. The server MUST reject any blob that is still referenced by another object with a blobHasReference SetError. If the blobId does not exist, the server MUST return a notFound SetError. Successfully destroyed blobIds are returned in a destroyed array. Failed destroys are returned in a notDestroyed map of blobId to SetError.

### 2.5. The "expires" response property

When this capability is present, the following blob creation responses MAY include an additional property: the resumableUploadUrl response, the Blob/upload response ([JMAP-BLOB], Section 4), and the Blob/convert response defined below.

* 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 or absent, the server provides no hint about the blob's lifetime.

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.

### 2.6. New method Blob/convert

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

Blob/convert performs server-side transformations on blobs. Like Blob/upload ([JMAP-BLOB], Section 4), 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

* unArchive: UnArchiveRecipe

* compress: CompressRecipe

* unCompress: UnCompressRecipe

* delta: DeltaRecipe

* patch: PatchRecipe

The response has the same structure as Blob/upload ([JMAP-BLOB], Section 4): a created map of creation id to an object containing id, type, and size 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 server MAY also return an expires property (see "The expires response property" above). Creation id backreferences (using the # prefix) resolve to this id 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.

#### 2.6.1. 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 supportedImageTypes capability.

* 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 supportedImageTypes, or the source blob is not a supported image format.

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

#### 2.6.2. ArchiveRecipe

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

* type: "String" The MIME type of the archive to create. MUST be one of the values in the server's supportedArchiveTypes capability. Defined values are "application/zip", "application/x-tar", and "application/x-cpio".

* 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.

##### 2.6.2.1. ArchiveEntry

An ArchiveEntry describes a single entry in an archive. It is used both as input (in ArchiveRecipe) and as output (in UnArchiveRecipe 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.

#### 2.6.2.2. 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.

#### 2.6.2.3. 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"]]
```

#### 2.6.2.4. 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.

### 2.6.3. UnArchiveRecipe

An UnArchiveRecipe 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 UnArchiveRecipe 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 supportedArchiveTypes.

* "tooLarge" — the source blob exceeds maxConvertSize.

### 2.6.4. 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 MIME type of the compression format to use. MUST be one of the values in the server’s supportedCompressionTypes capability. Defined values are "application/gzip", "application/x-bzip2", "application/x-xz", and "application/zstd".

* 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 supportedCompressionTypes.

* "tooLarge" — the source blob exceeds maxConvertSize.

#### 2.6.4.1. 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.

#### 2.6.4.2. 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.

#### 2.6.4.3. 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.

#### 2.6.4.4. 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.

### 2.6.5. UnCompressRecipe

An UnCompressRecipe 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 supportedCompressionTypes.

* "tooLarge" — the source blob exceeds maxConvertSize.

#### 2.6.6. 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 supportedPatchTypes 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 supportedPatchTypes.

* "tooLarge" — a referenced blob exceeds maxConvertSize.

##### 2.6.6.1. 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.

#### 2.6.6.2. 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.

#### 2.6.6.3. 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.

### 2.6.7. 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.

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

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

* "tooLarge" — a referenced blob exceeds maxConvertSize.

## 3. Examples

### 3.1. 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.

## 3.2. 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"]]
```

### 3.3. 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"]]
```

### 3.4. 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,
      "unCompress": {
        "blobId": "Bd81c7f",
        "type": "application/gzip"
      }
    },
    "u2": {
      "unArchive": {
        "blobId": "#u1",
        "type": "application/x-tar"
      }
    }
  }
}, "0"]]
```

The response for the UnArchiveRecipe 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.

### 3.5. 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"]]
```

# 4. Security Considerations

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

## 4.1. 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.

### 4.2. 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 UnArchiveRecipe 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.

### 4.3. Content Smuggling

The ability to split content into multiple blobs, recombine them via Blob/upload, 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.

## 5. IANA Considerations

### 5.1. JMAP Capability Registration for urn:ietf:params:jmap:blobext

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

Capability Name: urn:ietf:params:jmap:blobext

Intended use: common

Change Controller: IETF

Specification document: this document

Security and privacy considerations: this document, Security Considerations

### 5.2. JMAP Error Code Registrations

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

#### 5.2.1. 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

#### 5.2.2. 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 ([JMAP-BLOB], Section 5) to discover which objects reference the blob.

Reference: this document

## 6. 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-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

## 7. Acknowledgements

TODO

{backmatter}

## 8. Normative References

[HTTP-RESUMABLE-UPLOADS]
Kleidl, M., Zhang, G., and L. Pardue, "Resumable Uploads for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-resumable-upload-11, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-resumable-upload-11>.

[JMAP-BLOB]
Gondwana, B., Ed., "JSON Meta Application Protocol (JMAP) Blob Management Extension", RFC 9404, DOI 10.17487/RFC9404, August 2023, <https://www.rfc-editor.org/rfc/rfc9404>.

[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>.

[URI-TEMPLATE]
Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

## Author’s Address

Bron Gondwana
Fastmail
Email: brong@fastmailteam.com