WLCG Tape REST API (v1) reference document

Introduction

This document contains the WLCG tape REST API specifications created in collaboration by EOS+CTA (https://cta.web.cern.ch/cta/), dCache (https://www.dcache.org/) and StoRM (https://italiangrid.github.io/storm/).

The FTS project (https://fts.web.cern.ch/fts/) was also involved as the main client for the REST API.

In this document, the requirements level will be indicated following the RFC 2119 (https://datatracker.ietf.org/doc/html/rfc2119).

Main purpose of the WLCG tape REST API

The WLCG tape REST API offers a common interface allowing clients to manage disk residency of tape stored files and observe the progress of files as they are written to tape.

The protocol was designed to be implementation agnostic for the common tape-backed storage systems in WLCG: the same protocol is used to access a dCache, StoRM or EOS+CTA tape storage system.

Bulk operations provide an optimization to allow client and server to work efficiently with large numbers of files.

Set of operations provided by the API

Functionality	**Description**
STAGE	Request that tape-stored files are made available with disk latency.
RELEASE	Indicate that files previously staged through STAGE are no longer required with disk latency.
ARCHIVEINFO	Request information about the progression of writing files to tape

General information

Tape REST API endpoint identification

Each tape endpoint has a single, globally unique identifier. It is unspecified how an endpoint obtains this id or how global uniqueness is guaranteed.

Error message structure

All the error messages will use RFC 7807: https://datatracker.ietf.org/doc/html/rfc7807.

General error message example

```
{
  "status": 404,
  "title": "Not found"
}
```

JSON validation errors message example

```
{
  "detail": "paths - Field does not exist or is not a valid
non-empty array",
  "status": 400,
  "title": "JSON Validation error",
}
```

Fields description

Field name	Description	Field type	Optional
detail	A human-readable explanation specific to this occurrence of the problem.	String	Yes
status	The HTTP status code generated by the origin server for this occurrence of the problem.	Unsigned integer	No
title	A short, human-readable summary of the problem type	String	No
type	A URI reference that identifies the problem type This specification encourages that, when dereferenced, it provides human-readable documentation for the problem type (e.g., using HTML). When this member is not present, its value is assumed to be "about:blank".	String	Yes

Tape REST API discovery mechanism

The discovery mechanism allows the client to discover metadata about the service, including the endpoint.

Discovery endpoint

The discovery mechanism follows RFC 8615 (https://tools.ietf.org/html/rfc8615).

Clients MAY need to query the well-known endpoint of the Tape REST API that may be derived from the WebDAV endpoint.

They will do so by issuing a GET on the URI of the .well-known endpoint of the tape REST API.

URI of the .well-known endpoint of the tape REST API

The well-known endpoint is found by taking the WebDAV endpoint and resolving the absolute path */.well-known/wlcg-tape-rest-api*.

Conforming servers MUST support the discovery endpoint on all publicly advertised WebDAV endpoints.

Submission example

```
GET https://webdav.example.org:1234/.well-known/wlcg-tape-rest-api
```

Returned JSON response example

```
{
  "sitename": "cern-prod-tape-atlas",
  "description": "This is the CERN tape REST API endpoint for CTA ATLAS",
  "endpoints":[
       {
         "uri":"https://tape-api.example.org:1234/api/v1",
         "version":"v1",
         "metadata": {
         }
       }
  ]
}
```

Fields description

Field name	Description	Field type	Optional
sitename	The identifier (Tape REST API endpoint identification) for endpoint-targeted file level metadata.	String	No
description	Human readable description about the REST API endpoint	String	Yes
endpoints	Contains all the available tape REST API endpoints. There will be exactly one endpoint object per supported version of the REST API	Array	No
uri	The endpoint.	String	No
version	The version of the API that will be access via the uri	String	No
metadata	An extension point.	Object	Yes

Functionalities

STAGE

Description

A STAGE bulk-request requests that the storage starts the process of making the requested files available with disk latency.

Clients SHOULD only send tape-stored files. The server SHOULD reject all non tape-stored files with a file-specific error (disclosed during the stage polling). A non-tape stored file is anything that a tape storage system would refuse to store on tape such as directories or 0-length files. In case some files submitted already have a disk copy, the server MUST NOT reject them.

The client MAY include any per-file metadata for targeted endpoints. These metadata MUST be associated with some endpoint identifier (Tape REST API endpoint identification).

The endpoint for whom the metadata are not intended MUST ignore these metadata.

For each file submitted, the client MAY pass the time after which it is safe for the server to assume the file is no longer needed by this client. The server MAY ignore this information.

Submission of a stage bulk-request

Description

Allows the client to submit a set of files to be staged from tape.

A single request containing multiple files is logically equivalent to multiple requests each containing a single file.

Submission

Method	POST
Target	https://tape-api.example.org:1234/api/v1/stage
Content-Type	application/json

Submission JSON example

```
{
  "files": [
      {
         "path": "/experiment/test/test.txt",
         "targetedMetadata":
         {
           "sitename-from-well-known": {
              "activity": "exper_t0_daq"
           }
         }
      },
      {
         "path": "/eos/experiment/test/test2.txt",
         "diskLifetime":"PT1H",
         "targetedMetadata":
         {
            "sitename-from-well-known": {
              "activity": "exper_t0_daq"
            }
         }
      }
   ]
}
```

Fields description

Name	Description	Field type	Optional for the client	Remarks
files	The array of files to submit for staging	Array of File object	No	

File object description

Name	Description	Field type	Optional for the client	Remarks
path	The path of the file	JSON String	No	The server MUST sanitise the path (removal of duplicate slashes…)
diskLifetime	The duration after which the endpoint may assume the disk replica is no longer needed. The intended use is to allow the endpoint to recover from clients that "forgot" to release the file in a timely fashion.	JSON String ISO 8601 format	Yes	The endpoint MAY ignore this field. If the field is not provided then the endpoint MAY use a default value.
targetedMetadata	Allows the client to pass metadata to a targeted endpoint	Targeted metadata object	Yes	

Targeted Metadata

Targeted metadata allows clients to pass extra information for each file they want to stage. This field content is a JSON object that contains specific metadata for a targeted site name. The site is free to interpret this metadata or not and to act upon them.

Name	Description	Field type	Optional for the client	Remarks
`sitename-from-well-known`	The sitename to whom this metadata is intended for	JSON String	No	Sites to whom this metadata is not intended for MUST ignore it

Example of usage: If an experiment would like to pass activity information for each file to an EOS+CTA site at CERN, they can do so by filling the targeted metadata the following way:

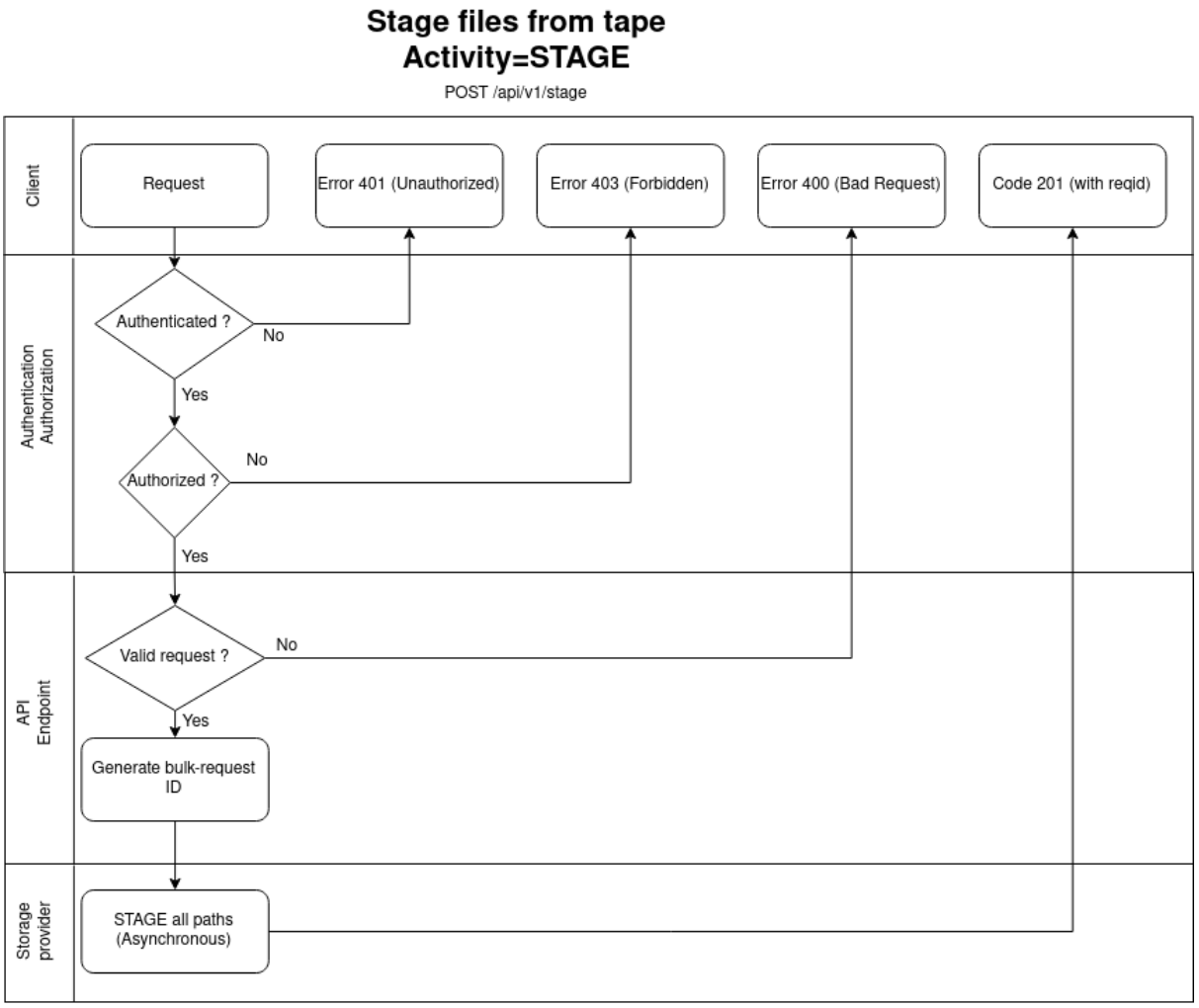
```
"targetedMetadata": {
  "atlas-eoscta@cern.ch": {
    "activity": "expe_t0_daq"
  }
}
```

The sitename can be found by querying the discovery endpoint of the REST API endpoint.

Response from the server example

```
< HTTP/1.1 201 CREATED
< Connection: Keep-Alive
< Location:
https://tape-api.example.org:1234/api/v1/stage/e2b8f2b0-9952-11ec-
9c53-fa163e0f6dc7
< Content-Length: 57
< Date: Tue, 01 Mar 2022 11:30:06 GMT
< Content-Type: application/type: json
<
{
  "requestId": "e2b8f2b0-9952-11ec-9c53-fa163e0f6dc7"
}
```

Processing expectations



Progress tracking of one request

Description

Allows the client to track the progression of a previously staged bulk-request.

The order of the files returned by the response is implementation-specific and is not guaranteed to be the same as the submission order.

Submission

Method	GET
Target	https://tape-api.example.org:1234/api/v1/stage/{stage_request_id}

Response from the server example 1

```
{
  "id": "93be38df-435c-4322-801d-b95e77ac5bbc",
  "startedAt":1646305456,
  "createdAt":1646305430,
  "files":[
    {
      "path":"/test/file.txt",
      "onDisk":true
    },
    {
      "path":"/test/file2.txt",
      "error":"Tape backend is unreachable",
      "onDisk":false
    }
  ]
}
```

Response from the server example 2


```
{
  "id": "93be38df-435c-4322-801d-b95e77ac5bbc",
  "completedAt":1646309000,
  "startedAt":1646305456,
  "createdAt":1646305430,
  "files":[
    {
      "path":"/test/file.txt",
      "finishedAt":1646306000,
      "startedAt":1646305470,
      "state":"COMPLETED"
    },
    {
      "path":"/test/file2.txt",
      "error":"Tape backend is unreachable",
      "finishedAt":1646305456,
      "startedAt":1646305456,
      "state":"FAILED"
    }
  ]
}
```

Response fields description

Name	Description	Field type	Optional for the server	Remarks
id	The id of the stage request.	JSON String	No	
createdAt	The time when the server received the request.	unsigned integer	No	
startedAt	the timestamp at which the first file belonging to the request has started	unsigned integer	No	There is no requirement for a server to support the concept of “starting”. If this concept is not supported, this timestamp will be equal to the createdAt value
completedAt	Indicates when the last file was finished	unsigned integer	Yes	
files	The files that got submitted and their state/disk residency	Array of File objects	No	

File object fields description

Name	Description	Field type	Optional for the server	Remarks
path	The path of the file	JSON String	No	The path MUST be unique among all the files returned in the response
finishedAt	The time at which the file request transitioned to one of the terminal states.	unsigned integer	Yes	if the field state is not set, this field MUST NOT be returned
startedAt	The timestamp at which the file request transitioned to STARTED	unsigned integer	Yes	if the field state is not set, this field MUST NOT be returned
error	The reason why a file could not be staged.	JSON String	Yes	If no error occured for the file, this field MUST not be returned
onDisk	Is true if the file is on disk, false otherwise	Boolean	Yes	If this field is set, the *state* field MUST not be returned
state	The State of processing this file	JSON String	Yes	If this field is set, the *onDisk* field MUST not be returned

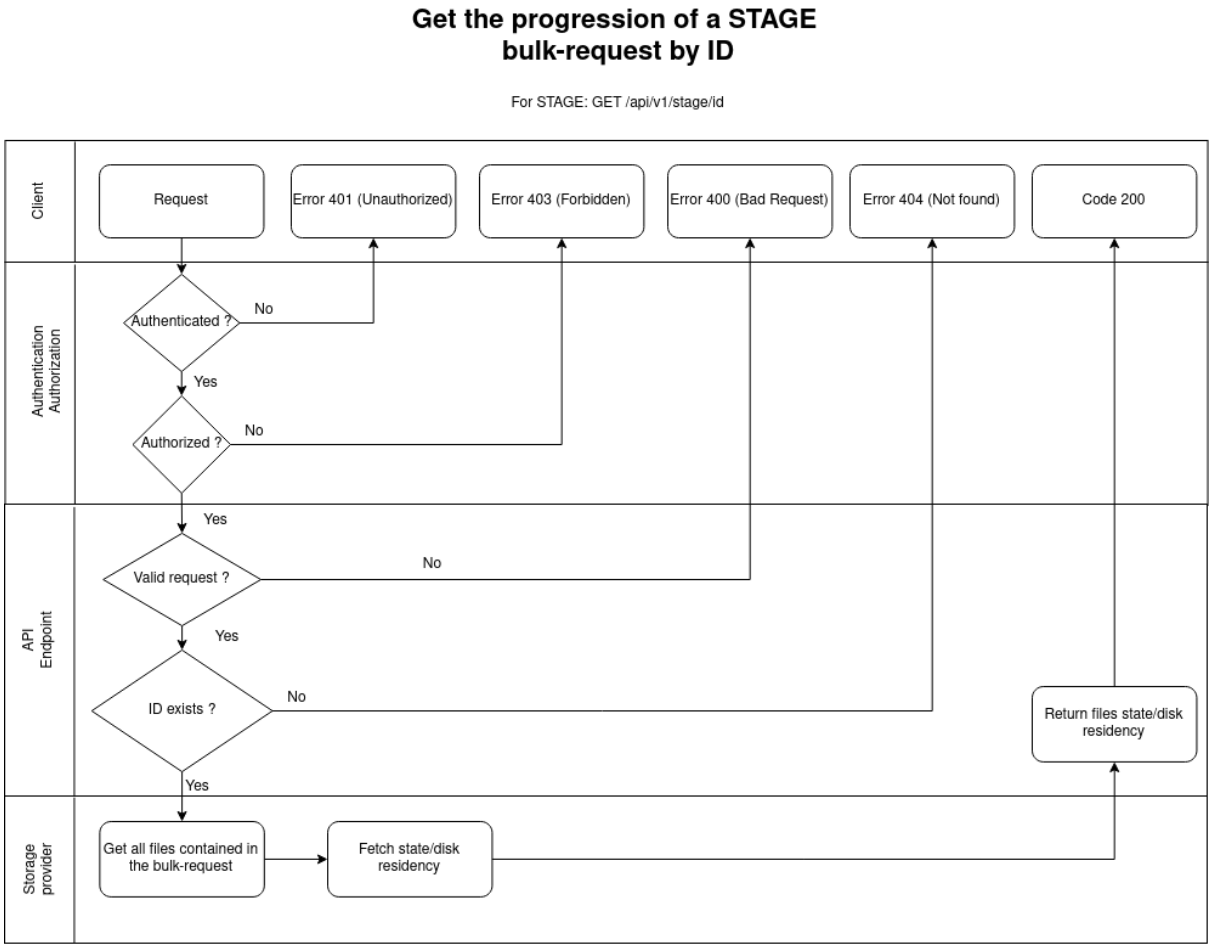
The different states of the file in the request

State	Semantic	Initial state	Terminal state
SUBMITTED	Stage processing has not started yet	X	
STARTED	Stage processing of this file has started		
CANCELLED	Stage has been cancelled by the client. The server provides no guarantee of disk-like latency.		X
FAILED	Stage processing is finished. The server was unable to provide file on disk		X
COMPLETED	Stage processing is finished, file is provided on disk with expected protection		X

Valid transitions for the state of the file

SUBMITTED → STARTED
STARTED → CANCELLED | FAILED | COMPLETED

Processing expectations



Cancellation

Description

Indicates that the targeted subset of files are no longer needed by the client.

For each cancelled file, the endpoint MAY cancel any staging activity. The endpoint MAY remove the disk replicas in case the files were successfully staged on disk.
The endpoint SHOULD take no action from a subsequent RELEASE request for any file that has been cancelled.

If at least one file in the targeted subset does not belong to the initially submitted stage request, the endpoint MUST return an error to the client and take no action from the request.

Submissionhttps://github.com/

Method	POST
Target	https://tape-api.example.org:1234/api/v1/stage/{stage_request_id}/cancel
Content-Type	application/json

Submission JSON example

```
{
  "paths":[
    "/path/example.txt",
    "/path/example2.txt"
  ]
}
```

Fields description

Name	**Description**	**Field type**	**Optional for the client**	**Remarks**
paths	The array of file paths to cancel	Array of String	No	The paths provided MUST belong to the initially submitted STAGE request

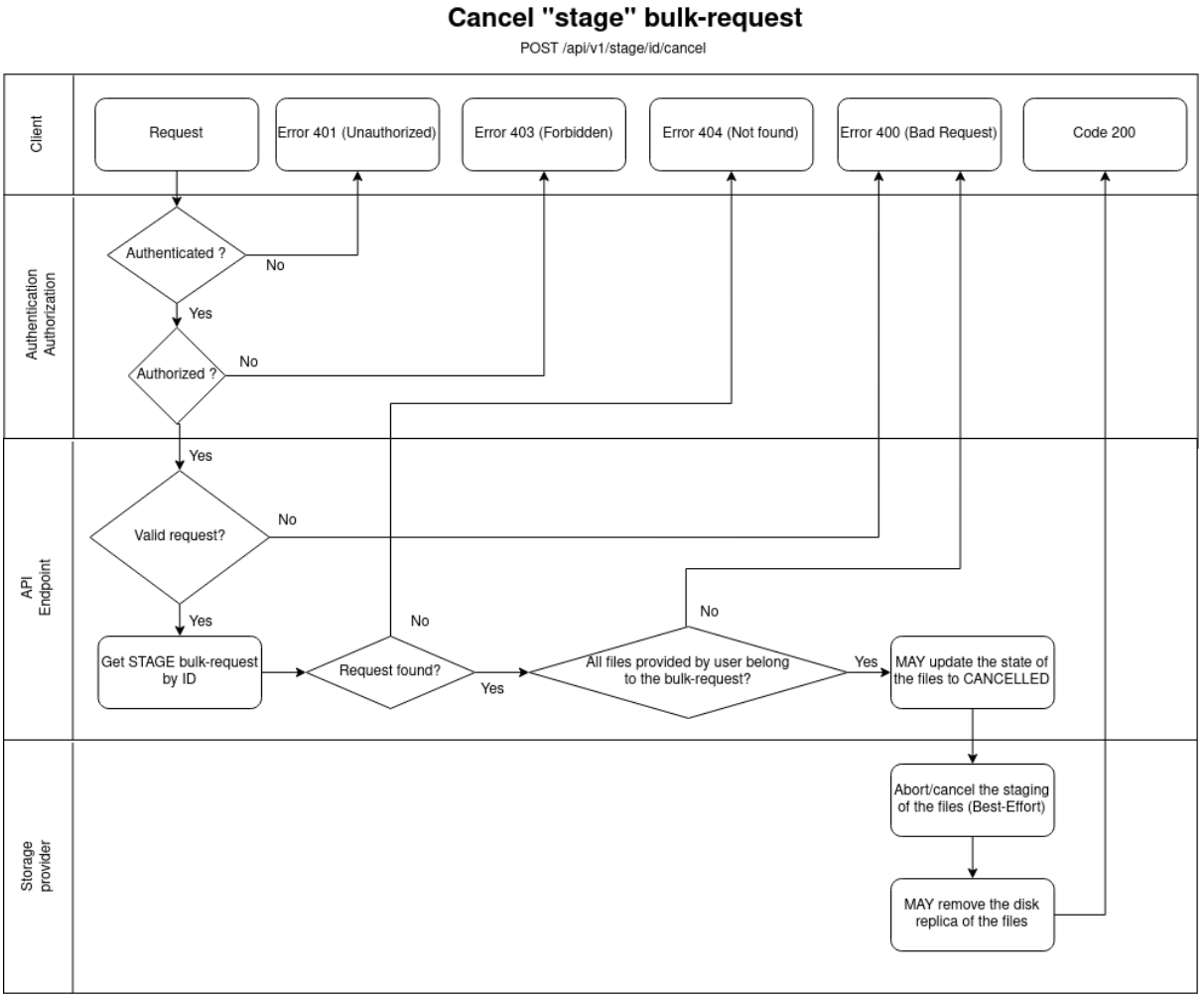
Response from the server example

```
< HTTP/1.1 200 OK
< Connection: Keep-Alive
< Content-Length: 0
< Date: Thu, 03 Mar 2022 10:46:24 GMT
```

Response from the server if one file does not belong to the stage request example

```
{
  "detail": "The file /file/example.txt does not belong to the
STAGE request f4813a9c-9ade-11ec-9bc0-fa163e0f6dc7. No
modification has been made to this request.",
  "status": 400,
  "title": "File missing from stage request"
}
```

Processing expectations



Deletion of one request

The deletion of a previously submitted STAGE bulk-request will trigger a **cancellation** (best-effort) of all the files that belong to the stage request. Once deleted, the client will no longer have access to this request.

Submission

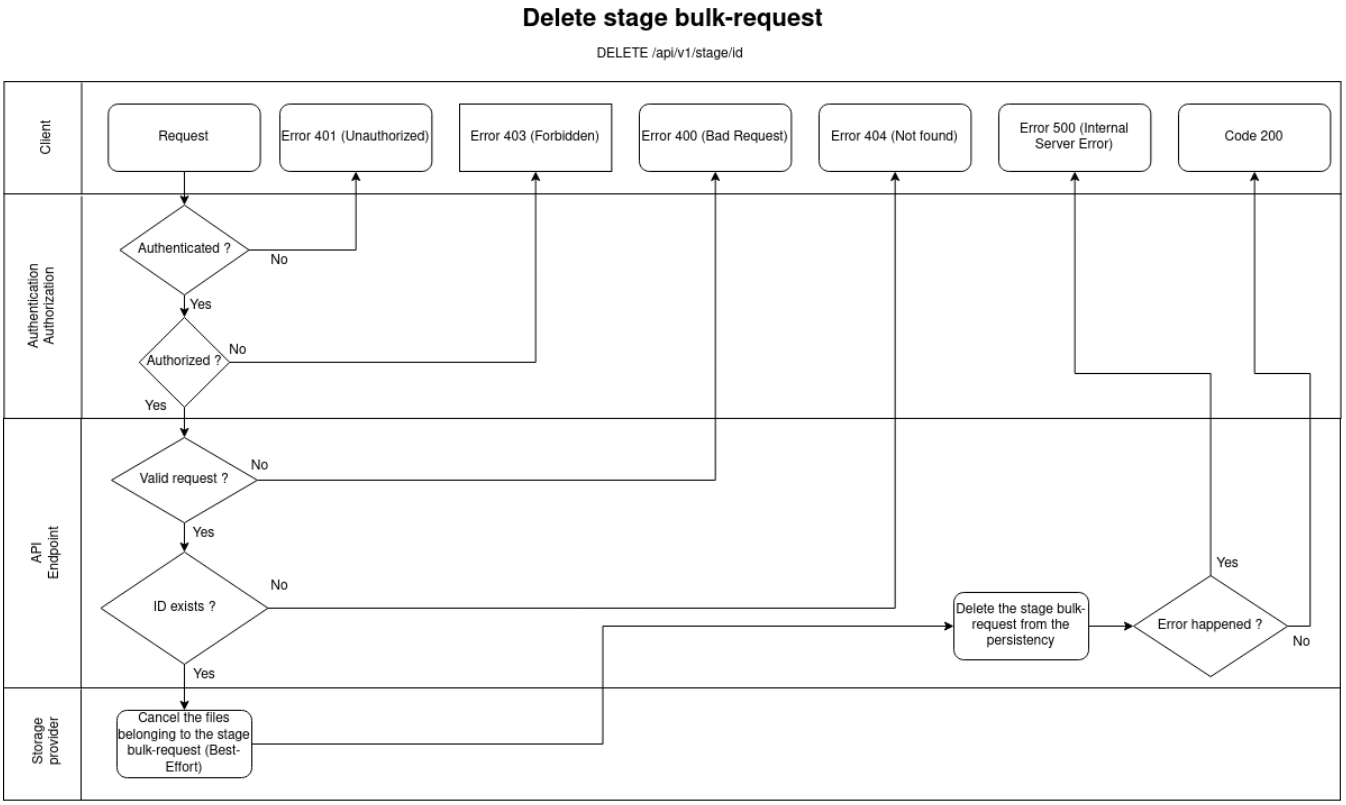
Method	DELETE
Target	https://tape-api.example.org:1234/api/v1/stage/{stage_request_id}

Response from the server example

< HTTP/1.1 200 OK
< Connection: Keep-Alive
< Content-Length: 0
< Date: Thu, 03 Mar 2022 10:46:24 GMT

r

Processing expectations



RELEASE

Description

Allows the client to indicate that they no longer require disk-like latency for the staged files.

The storage provider MAY remove these files from the disk cache.

The client MUST provide the stage request id in the URL of their request.

Submission

Method	POST
Target	https://tape-api.example.org:1234/api/v1/release/{stage_request_id}
Content-Type	application/json

Submission JSON example

```
{
  "paths":[
    "/path/example.txt",
    "/path/example2.txt"
  ]
}
```

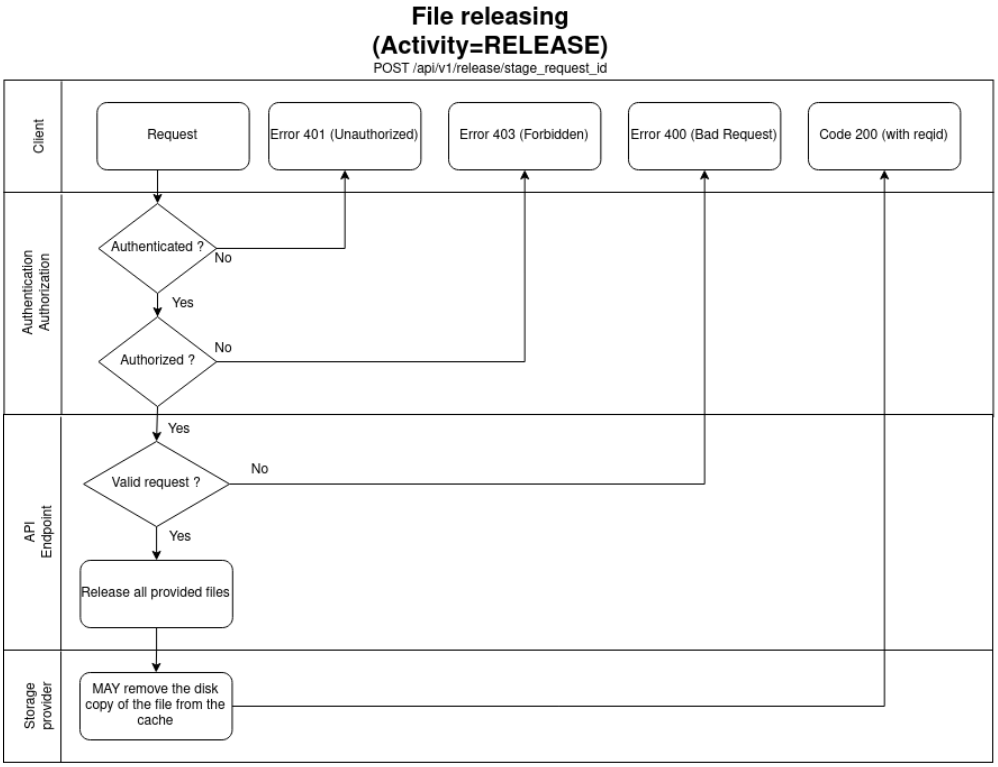
Fields description

Name	**Description**	**Field type**	**Optional for the client**	**Remarks**
paths	The array of file paths to release	Array of String	No	

Server response example

< HTTP/1.1 200 OK
< Connection: Keep-Alive
< Content-Length: 0
< Date: Thu, 03 Mar 2022 10:46:24 GMT

Processing expectations



ARCHIVEINFO

Description

Request information about the progress of writing files to tape.

If the client asks for too many files, the server MUST NOT return an error. Instead it will only return the response about the files it could fetch.

Submission

Method	POST
Target	https://tape-api.example.org:1234/api/v1/archiveinfo
Content-Type	application/json

Submission JSON example

```
{
  "paths":[
    "/path/example.txt",
    "/path/example2.txt"
  ]
}
```

Fields description

Name	**Description**	**Field type**	**Optional for the client**	**Remarks**
paths	The array of file paths to get residency information	Array of String	No	

Response from the server example

```
[
  {
    "locality": "DISK_AND_TAPE",
    "path": "/eos/ccaffy/tape/test.txt"
  },
  {
    "error": "USER ERROR: file does not exist or is not accessible
to you",
    "path": "/file/does/not/exist"
  }
]
```

Response fields description

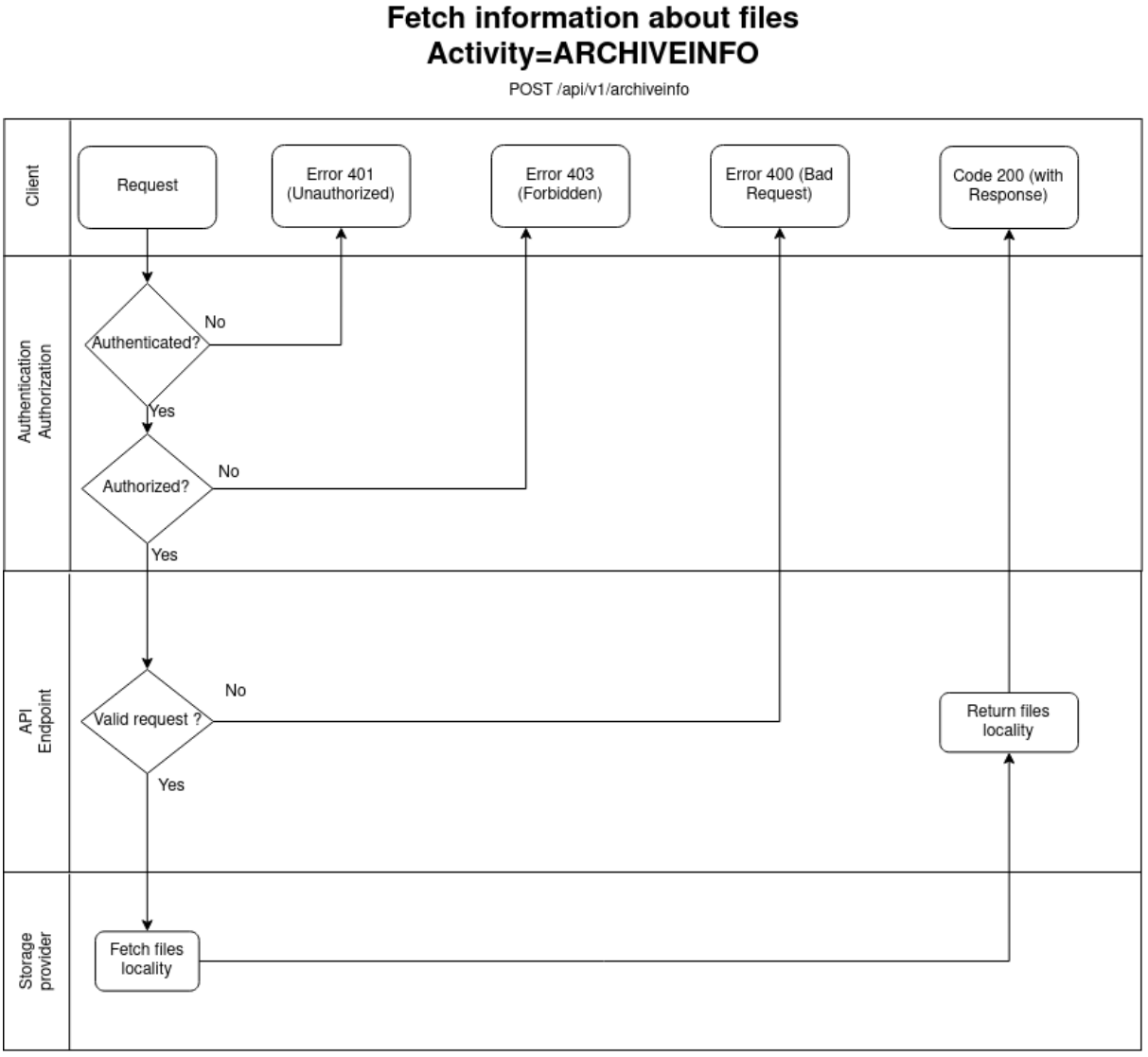
Name	**Description**	**Field type**	**Optional for the server**	**Remarks**
locality	The file locality	Locality	Yes in the case the locality could not be fetched	If a provided file does not exist, the locality MUST NOT be returned.
path	The path of the file	JSON String	No	
error	The error message is an indication that the file is unlikely going to be written to tape during the request lifetime. It allows the user to fail the transfer fast.	JSON String	Yes	An archive request lifetime is around 24h

Locality object description

This enumeration was strongly inspired from the SRM protocol TFileLocality object (https://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html#_Toc241633052).

Locality	**Semantic**
DISK	Indicates that the file's data is located only on disk storage.
TAPE	Indicates that the file's data is located only on a tape storage system.
DISK_AND_TAPE	Indicates that the file's data is located on a tape as well as on a disk storage system.
LOST	Indicates the file's data is considered unrecoverable.
NONE	May be used if the file is empty (0 size)
UNAVAILABLE	Indicates that the file's data is unavailable due to a temporary hardware failure, disk server being rebooted, etc.

Processing expectations



Authentication mechanisms

The endpoint will support:

- X509 + VOMS
- VO issued tokens (WLCG JWT tokens)

Some design principles

Storage-agnosticity of the clients

The client-side of the API should remain storage-agnostic: they should not adapt their messages depending on the targeted endpoint.

Endpoints MUST NOT refuse site-specific additional information for staging

The endpoint MUST NOT refuse a staging request if it contains site-specific additional information. The endpoint MUST ignore any such additional information.

New requirement on one site should not affect the others

The fast deployment of a bug fix or new requirement on one site should not affect the other sites.

Ideas for future versions of the API

During the development of this protocol, a number of ideas were discussed that were identified as non-critical. In order to facilitate the convergence of this protocol, those ideas were recorded as input for some future version of this protocol.

Limitations hints for the user

Provide some mechanism through which a client may learn to what extent the on-going requests are reaching some internal limits on the server's ability to stage files concurrently.

- In flight staging at the same time
- Max number of entries in a bulk request
- Minimum time between two polls (GET)
- Maximum time for which we will keep the bulk request on our system (automatic cleaning). Defined by the client and this value will have a maximum allowed.

Second .well-known access point

We might provide a second ".well-known" access point. It will contain the limitations we would like to impose on the user (Limitations to impose to the user).

Advertise targeted metadata audiences in .well-known

Advertise metadata audiences/targets in the .well-known mechanism so that clients (FTS for example) can filter before sending metadata from the experiments.

Between two stage progress polling, only send information about the files that were successfully staged

In a future version of the API, we can ask the client to provide a flag allowing us to know whether it is stateful or not. A stateful client will keep track of the files that are already staged and will therefore not be interested in getting information about them anymore. The server will then only send information about the files that were not staged yet.

Notification of file flushing and staging activity

The current API model uses polling to determine when some activity has completed: either a file being written to tape or a stage request finishing. Polling has the disadvantage of forcing compromise on the polling frequency: too often places unnecessary stress on the storage system, too infrequently incurs a delay in learning of state changes. An event-based model would avoid such compromises, allowing a client to learn of changes quickly without stressing a storage system.

The end

Items for Review

This section should host items raised up for review, either upon careful reading of the document or upon implementation challenges.

Staging request (Mihai)

1. Is the order of the items in the staging request preserved during the subsequent polling?

a. If the order is not preserved:
    - Should the client use the path as the key to items in the response?
    - What happens if the client sends the same path multiple items (with or without different file metadata)? Will the server refuse the request or deduplicate the paths?

b. If the order is preserved:
    - Do the servers guarantee they will return the same number of items in the polling response? (If that's the case, the client must refuse a polling response whose number_of_files_in_reply != expected_number_of_files)

2. Sanitizing “paths” field in the staging request. In this particular case, the concrete sanitize example is “collapsing consecutive slashes (e.g.: “//path/file” → “/path/file”)
    - Should the client send a sanitized request and the server refuse it if not-sanitized?
    - Will the server accept non-sanitized input, but respond with a sanitized version during polling, “/cancel” and “/release” endpoints?
    - Must the server use the exact same “path” as used by the client during the initial request?
3. Can the server respond with an error as soon as a path is not valid? Or should it validate all of them and return an error comprising all the single errors?

.well-known (Mihai)

1. The Tape REST API protocol must clearly define the format of the version field.
    a. Example: would this include only integers (v1) or also subversions (v1.2). Do we include additional descriptors? (v1.2-alpha)