# docuteam bridge

## documentation for client applications

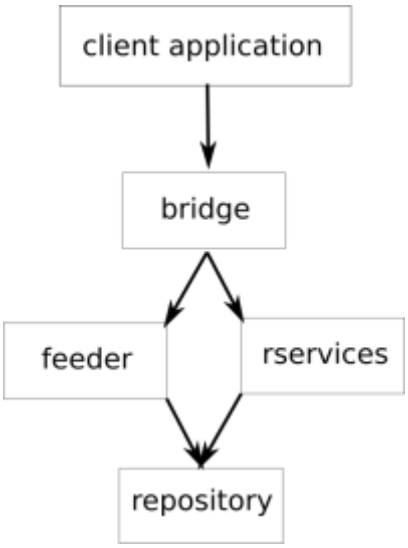
Documentation for docuteam bridge v.1.0.0

## goal

Client applications should be able to submit a deposition (data and metadata) to our ingest platform. Depositions are picked up by docuteam feeder workflows, usually processing and storing the information in a repository. After a successful ingest, feeder (and subsequently bridge) return PIDs for every object (file, folder) within the deposition. Using these PIDs, clients are able to access (read and change) the deposited objects.

In the Open Archival Information System (OAIS) terminology:

- the bridge deposition API recieves Submission Information Package (SIP)
- after preservation actions, the SIP are ingested into the repository by feeder and become Archival Information Package (AIP)
- the bridge access API enables client application to retrieve Dissemination Information Package (DIP)




## key points

- Bridge is a set of rest APIs that respond in JSON (and binary data)
- Bridge is agnostic to package format
  - Use the simple, bagit-based format docuteam dublin core 1.0 (see SIP docuteam dublin core1.0)
  - Use Matterhorn METS (see specification)
  - Use other formats like e. g. eCH-0160 or SEDA.

- Bridge is composed of 3 APIs:
  - **depositions**: deposition of new packages
  - **access**: read data successfully deposited in the repository
  - **changes**: update or purge repository objects

# bridge APIs

For security reasons, bridge enforces HTTPS. Access is restricted via tokens that must be transmitted with each request via using the “token” parameter (regardless the HTTP method: GET, POST, PUT, PATCH, DELETE). Roles are associated to tokens and limit their scope of operation.

- **depositions**
  - new depositions are temporarily stored by bridge. They are made available to feeder, which in turn processes and stores them into the repository,
  - depositions are identified by bridge IDs, that are returned with the HTTP response to a deposition creation request,
  - upon successful archiving in the repository,
    - the status of the deposition is set to „archived“
    - PIDs, which are the repository’s Persistent IDs are made available by bridge. PIDs are required to access (read, modify, purge) repository objects.
    - Each digital object in the repository (folder, file) gets its own PID and relations between the customer's application IDs and PIDs are explicit in the deposition metadata
      - the SIP is deleted from bridge as the information is preserved by the repository
      - **changes** target (for update or purge) one single object in the repository identified by its PID. They can be limited to:
      - metadata only (data is unchanged, only for updates)
      - data only (metadata is unchanged)
      - object (metadata+data are modified)
      - **access**
      - gives access to the full DIP corresponding to a deposited SIP,
      - or gives access to metadata, data or on-the-fly converted formats of specific repository objects,
      - this API is a proxy to docuteam rservices (a component offering high level access to Fedora objects, such as DIP and preview generation)

## 0 - authentication and roles

Bridge relies solely on tokens for authentication an authorization. Tokens are bound to institutions and roles and restrict the API usage on that basis.

- An authentication token must be at least 15 characters long.
- Tokens are passed via the “token” HTTP parameter, for example when using GET, this results in urls of the form:

```
http://server/access/sync_original/:pid?token=123456789012345
```

### roles

There are 5 roles:

- The 3 first roles are limited to the organization they are bound to:
  - **read**: is restricted to the access API
  - **create**: has the same authorizations as read, and can in addition list and create depositions via the deposition API
  - **manage**: has the same authorizations as read, and can in addition update or purge repository objects via the nodes API
- The 2 last roles are not limited to any organization
  - **admin**: authentication and authorization management, i.e. token administration via the API or the GUI
  - **feeder**: super user, can do anything except token administration, including depositions status updates

### recommendation

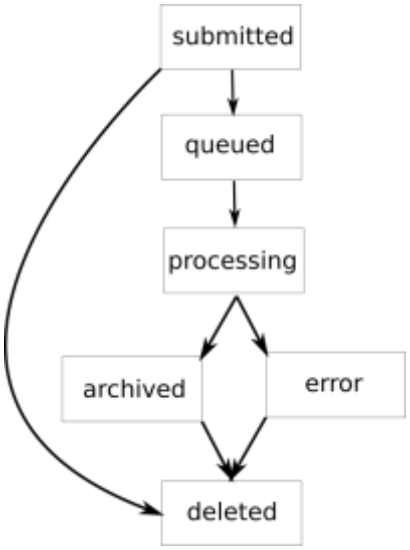
- use only one token with read roles in applications that do not require to create nor update depositions,
- use only one token with create role in application that make use of the deposition API but not the changes API,
- use only one token with manage role in applications that make use the changes API.

## 1 - depositions API

### deposition statuses

- *submitted*: the deposition was received by bridge
- *queued*: the deposition has been attributed to a queue in feeder
- *processing*: the deposition has been downloaded by feeder, which is processing it (preservation actions and storage in the repository)
- *archived*: the deposition was successfully processed by feeder and archived into the repository. Persistent identifiers allocated to the deposition's objects are available in bridge and the binary object has been deleted from bridge.
- *error*: something went wrong during the deposition's processing by feeder, see “message” fields in response)
- *deleted*: the deposition was deleted from bridge (not from the repository!)

The deposition status can be managed by the role “feeder”. Other non-reader roles may only delete depositions (set the status to deleted), except when status is set to “processing”.




**deposition responses**

Responses are given in JSON or a binary format. Each JSON response is a list of depositions metadata matching the query. Generic structure:

```
{ "api" :
   { "name": "docuteam bridge",
     "version": 1.0.0 },
 "response" :
   [
     { "id": 1234,
       "uploaded_at": "2018-11-03T11:13:39.278026Z",
       "queued_at": "2018-11-03T14:16:12.678056Z",
       "processed_by_feeder_at": "2018-11-03T14:16:12.678016Z",
       "archived_at": "2018-11-03T14:16:12.678016Z",
       "deleted_at": null,
       "status": "archived",
       "feeder_response": { json-blackbox },
       "organization": "museumplus",
       "repository_key": "museumplus",
       "package_format" : "DocuteamDublinCore1.0",
       "package_attached" : true,
       "package_byte_size": 2716786
     }
   ]
 "request" :
   { "organization": "museumplus",
     "role": "reader",
     "requested_at": "2018-11-03T11:13:39.278026Z"}
}
```

Key elements include:

- “id” is the deposition identificater, in other words bridge internal reference to depositions. It is notably required to access a specific deposition,
- “status” is the deposition status, as described above,

- “feeder_response” is also formatted in JSON. It is a black box from bridge's perspective. Upon deposition success, feeder will return a structure of the form:

```
{ "pids":
    [
      { "clientId":"c1", "pid":"CH-1234565-7:1"},
      { "clientId":"c2", "pid":"CH-1234565-7:2"},
         ...
    ],
  "feeder_version": "5.4.0"
}
```

It must be noted that:

- the „clientId“ corresponds to the mandatory id's submitted by the client application for each object within the SIP (for example in the case of Docuteam DublinCore SIP it is located in dc.xml files and using the following syntax *<dc:identifier>clientId:d4FTw3v6T</dc:identifier>* ).
- the „pid“ is the persistent identifier allocated by the repository. It is of the form namespace:id where namespace is generally the institutional ISIL code (for example: CH-1234565-7:2)

## deposition routes

### deposition routes overview

```
GET    /depositions       depositions#index
POST   /depositions       depositions#create
GET    /depositions/:id   depositions#show
PATCH  /depositions/:id   depositions#update
PUT    /depositions/:id   depositions#update
```

### deposition routes detailed

| Action | Description | Examples |
|---|---|---|
| **create** | HTTP POST on depositions | `curl -X POST -F "package=@sip.zip" "https://bridge-stage.docuteam.ch/depositions?token=1234567890123456&package_format=DocuteamDublincore1.0"` |
| | parameter: `package_format`, constrain: binary data is mandatory | |
| **list** / **show** | **HTTP GET** on /depositions, **parameters:** - id (optional, deposition id), -status (optional), -from (optional, format:YYYY-MM-DD), -until (optional, format:YYYY-MM-DD), -organization (optional) | `curl "https://bridge-stage.docuteam.ch/depositions?token=123456789012345"`<br>`curl "https://bridge-stage.docuteam.ch/depositions?token=123456789012345&id=2"`<br>`curl "https://bridge-stage.docuteam.ch/depositions?token=123456789012345&status=submitted"`<br>`curl "https://bridge-stage.docuteam.ch/depositions?token=123456789012345&status=submitted&organization=customerx"`<br>`curl "https://bridge-stage.docuteam.ch/depositions?token=123456789012345&from=2018-11-01&until=2018-11-30"`<br>`curl "https://bridge-stage.docuteam.ch/depositions?token=123456789012345&status=error&from=2018-11-01"` |
| **retrieve binary data** | **HTTP GET** on /depositions/:id returns SIP | `curl "https://bridge-stage.docuteam.ch/depositions/1?token=123456789012345" --output sip.zip` |
| **update** | **HTTP PUT** on /depositions/:id **parameters:** - id (deposition*id)* - *status (values: deleted, processing, ...)* - *feeder*response (url encoded string) | `curl -X PUT "https://bridge-stage.docuteam.ch/depositions/12345?token=123456789012345&status=deleted"`<br>`curl -X PUT "https://bridge-stage.docuteam.ch/depositions/23?token=12super34token56&status=archived&feeder_response=%7B%22pids%22%3A%5B%221%22%2C%222%22%5D%7D"` |

## 2 - access API

This API is a of proxy to docuteam rservices. Rservices is offers high level access to Fedora objects. For example, it is able to generate DIP starting from the top level object of an archival package and assembling it recursively. Another notable feature is the the on-the-fly generation of thumbnails and, more generally, format migrations.

Bridge adds an authentication and authorization layer on top of rservices.

This API retrieves data from the archive, hence expects PIDs (and not bridge internal IDs, as it is the case for the depositions API).

In version 1.0.x, bridge is limited to synchronous requests, meaning that the required object is prepared and returned at once. In other words, it is not yet possible to ask for the generation of a DIP and to come back later to download it.

#### access routes overview

```
GET   /access/sync_preview/:pid    sync_preview#download
GET   /access/sync_original/:pid   sync_original#download
GET   /access/sync_dip/:pid        sync_dip#download
GET   /access/sync_metadata/:pid   sync_metadata#download
```

#### access routes detailed

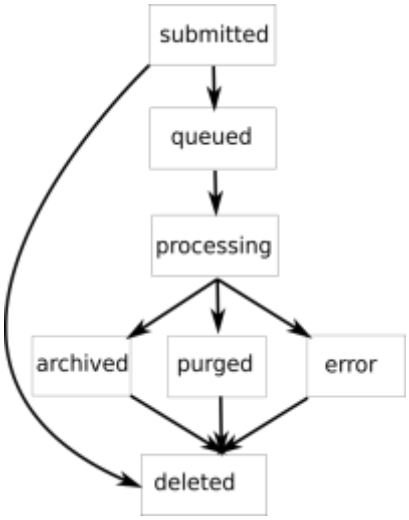
| Action | Description | Examples |
|---|---|---|
| **dip** | **HTTP GET** on /access/sync*dip/:pid will generate a MatterhornMets DIP, **parameters:** - recursively=(true or false, default is false), - verifyChecksum=(true or false, default is false) \| <code>curl „https://bridge-stage.docuteam.ch/access/sync_dip/CH-123456-7:38?token=123456789012345“ –output dip.zip</code> <code>curl „https://bridge-stage.docuteam.ch/access/sync_dip/CH-123456-7:38?token=123456789012345&recursively=true –output dip.zip</code> \| \| **original** \| **HTTP GET** on /access/sync*original/:pid, sends back the original file from the archive | `curl "https://bridge-stage.docuteam.ch/access/sync_original/CH-123456-7:38?token=123456789012345" --output file.pdf` |
| **preview / file migration** | **HTTP GET** on /access/sync*preview/:pid, compute on-the-fly and returns a file migration of archived file according to rservices migration settings \| <code>curl „https://bridge-stage.docuteam.ch/access/sync_original/CH-123456-7:38?token=123456789012345“ –output file.pdf</code> \| \| **metadata** \| **HTTP GET** on /access/sync*metadata/:pid, returns the EAD metadata of object | `curl "https://bridge-stage.docuteam.ch/access/sync_original/CH-123456-7:38?token=123456789012345" --output ead.xml` |

## 3 - changes API

Changes are similar to depositions, however they target only on specific existing object in the repository in order to replace or purge it. The targeting is done via the object's PID.

### changes statuses

- *submitted*: a new change corresponding to an update or purge was created in bridge
- *queued*: the change has been attributed to a queue in feeder
- *processing: the change has been downloaded by feeder, which is processing it* * archived: the change was successfully processed, meaning the object in the repository was updated (depending on the repository setting this may create a new version of the object or replace it)
- *purged*: the object was successfully purged from the repository
- *error*: something went wrong, see the “message” field itself located in „feeder_response“ within the JSON
- *deleted*: the change was deleted from bridge




### changes responses

The changes JSON responses are similar to the deposition responses, but have two additional fields:

- “pid” that relates to the repository targeted id in the archive
- “task” that describes the action performed (update, purge)

Practically, changes responses look like this:

```
{ "api":
  { "name": "docuteam bridge",
    "version": 1.0.0 },
"response":
  [
    { "id": 4321,
      "uploaded_at": "2018-11-03T11:13:39.278026Z",
      "queued_at": "2018-11-03T14:16:12.678560Z",
      "processed_by_feeder_at": "2018-11-03T14:16:12.678016Z",
      "archived_at": "2018-11-03T14:16:12.678016Z",
      "purged_at": null,
      "deleted_at": null,
      "status": "archived",
      "feeder_response": { json-blackbox },
      "organization": "myorganisation",
      "repository_key": "myrepository",
      "package_format" : "DocuteamDublinCore1.0",
      "package_attached" : true,
      "package_byte_size": 2716786,
      "task" : "node_update",
      "pid" : "CH-654321-0:87654"
     }
  ]
"request":
  { "organization": "myorganiztion",
    "role": "manage",
    "requested_at": "2018-11-03T11:13:39.278026Z"}
```

```
}
```

## changes routes

### changes routes overview

```
POST   /changes/:id   changes#update
GET    /changes       changes#index
GET    /changes/:id   changes#show
PUT    /changes/:id   depositions#update
```

### changes routes detailed

| Action | Description | Examples |
|---|---|---|
| **create** | **HTTP POST** on /changes/:id,<br>**parameters:**<br>- pid (repository persistent identifier)<br>- package*fromat (default: MatterhornMets)*<br>- *task (data*update, metadata*update, object*update or data*delete, object*delete) | `curl -X POST -F "package=@sip.zip" "https://bridge-stage.docuteam.ch/changes ?token=12345678901234 5&pid=CH-654321-0:3&task=object_update&package_format=DocuteamDublincore1.0"`<br>`curl -X POST "https://bridge-stage.docuteam.ch/changes/CH-654321-0:3?token=12345678901234 5&task=object_delete"` |
| **list / show** | **HTTP GET** on /changes,<br>**parameters:**<br>- id (optional, change id),<br>-status (optional),<br>-from (optional, format:YYYY-MM-DD),<br>-until (optional, format:YYYY-MM-DD),<br>-organization (optional) | `curl "https://bridge-stage.docuteam.ch/changes?token=123456789012345"`<br>`curl "https://bridge-stage.docuteam.ch/changes ?token=123456789012345&id=2"`<br>`curl "https://bridge-stage.docuteam.ch/changes?token=123456789012345&status=submitted"`<br>`curl "https://bridge-stage.docuteam.ch/changes ?token=123456789012345&status=submitted&organization=customerx"`<br>`curl "https://bridge-stage.docuteam.ch/changes?token=123456789012345&from=2018-11-01&until=2018-11-30"`<br>`curl "https://bridge-stage.docuteam.ch/changes?token=123456789012345&status=error&from=2018-11-01"` |
| **retrieve binary data** | **HTTP GET** on /changess/:id returns SIP | `curl "https://bridge-stage.docuteam.ch/changes/1 ?token=123456789012345" --output sip.zip` |
| **update** | **HTTP PUT** on /changes/:id<br>**parameters:**<br>- id (change*id)*<br>*- status (values: deleted, processing, ...)*<br>*- task (data*update, metadata*update, object*update or data*delete, object*delete)<br>- feeder*response (url encoded string)*\| <code>curl -X PUT „*https://bridge-stage.docuteam.ch/changes/12345 ?token=123456789012345&status=deleted*“</code> <code>curl -X PUT „*https://bridge-stage.docuteam.ch/changes/23 ?token=12super34token56&status=archived&feeder*response=%7B%22pids%22%3A%5B%221%22%2C%222%22%5D%7D“</code> | |