o2r web API documentation

Current version of the API: `v1`

About

The o2r web API acts as the interface between the o2r microservices and the web interface.

The API provides services around the executable research compendium (ERC), or "compendium" for short, which is documented **in the ERC spec**.

General notes

The API is implemented as a RESTful API. The entrypoint for the current version is `/api/v1`.

Unless specified otherwise, responses are always in JSON format. Body parameters in `POST` requests are expected in `multipart/form-data` format. Requests to the API should always be made with a secure connection using `HTTPS`. Some requests require authentication with a specific user level.

License

Next

View compendium

List compendia

Returns up to 100 results by default.

`curl https://…/api/v1/compendium?limit=100&start=2`

`GET /api/v1/compendium?limit=100&start=2`

```
200 OK

{
  "results": [
    "nkm4b",
    "asdis",
    "nb2sm",
    …
  ]
}
```

You can also filter the results.

- Filter by `user`:
  - `curl https://…/api/v1/compendium?user=0000-0002-1825-0097`
  - `GET /api/v1/compendium?user=0000-0002-1825-0097`
- Filter by `doi`:
  - `curl https://…/api/v1/compendium?doi=10.9999%2Ftest`
  - `GET /api/v1/compendium?doi=10.9999%2Ftest`

```
200 OK

{
  "results": [
    "nkm4b",
    "nb2sm"
  ]
}
```

If there is no compendium found, the service returns an empty list.

`GET /api/v1/compendium?doi=not_a_doi`

```
200 OK

{
  "results": []
}
```

URL parameters for compendium lists

- `job_id` - Comma-separated list of related job ids to filter by.
- `user` - Public user identifier to filter by.

- `doi` - A DOI to filter by.
- `start` - Starting point of the result list. `start - 1` results are skipped. Defaults to `1`.
- `limit` - Limits the number of results in the response. Defaults to `100`.

View single compendium

This includes the complete metadata set, related job ids and a tree representation of the included files. The `created` timestamp refers to the upload of the compendium. It is formated as ISO8601.

`curl https://…/api/v1/$ID`

`GET /api/v1/compendium/:id`

```
200 OK

{
  "id":"comid",
  "metadata": … ,
  "created": "2016-08-01T13:57:40.760Z",
  "files": …
 }
```

URL parameters for single compendium view

- `:id` - the compendiums id

Error responses for single compendium view

```
404 Not Found

{"error":"no compendium with this id"}
```

List related execution jobs

`curl https://…/api/v1/compendium/$ID/jobs`

`GET /api/v1/compendium/:id/jobs`

```
200 OK
{
  "results": [
    "nkm4L",
    "asdi5",
    "nb2sg",
    …
  ]
}
```

If a compendium does not have any jobs yet, the returned list is empty.

```
200 OK
{
  "results": [ ]
}
```

URL parameters for related execution jobs

- `:id` - compendium id that the results should be related to

Previous   Next

Candidate process

After uploading a compendium is *not* instantly publicly available. It is merely a **candidate**, because metadata still must be completed for the compendium to be valid.

The following process models this intermediate state of a compendium.

Creation and view

Candidates can be identified by the property `candidate`. It is set to `true` after creating a new compendium by upload or public share submission **and** the authoring user having reviewed the metadata.

Note

It is not possible to circumvent the metadata review. Only a successful metadata update can set `candidate: true`.

Example:

```
{
  "id":"12345",
  "metadata": … ,
  "created": "2016-08-01T13:57:40.760Z",
  "files": …,
  "candidate": true
}
```

Only the creating user and users with required level can view a candidate and see the `candidate` property while it is `true`.

When accessing a list of compendia for a specific user *as that user*, then this list is extended by available candidates. The candidates may be added to the response independently from any pagination settings, i.e. if a client requests the first 10 compendia for a user having two candidates, the client should be prepared to handle 12 items in the response.

Metadata review and saving

After the user has reviewed and potentially updated the metadata as required and saved them successfully, then the candidate status is changed (`candidate: false`) and the compendium is publicly available.

The `candidate` property is not exposed any more if it is `false`.

It is *not* possible to save invalid metadata or to manually change the `candidate` property, therefore a compendium cannot become a candidate again after successful completion of the creation.

Deletion

Unlike published compendia, a candidate can be deleted by a the authoring user, see delete compendium.

Compendium file listing

The file listing is returned in the single view of a job or compendium. It includes the complete content of the bagtainer in its current state. If a job has been run and the programme outputs new data, this new data is included as well.

File listings are represented as a Object. The file structure for a synthetic job `nj141` is as follows.

```
nj141
├── bagit.txt
└── data
    ├── paper.Rmd
    └── Dockerfile
```

is be represented as

```
{
  "path": "/api/v1/job/nj141/data",
  "name": "nj141",
  "children": [
    {
      "path": "/api/v1/job/nj141/data/bagit.txt",
      "name": "bagit.xt",
      "type": "text/plain",
      "size": 55
    },
    {
      "path": "/api/v1/job/nj141/data/data",
      "name": "data",
      "children": [
        {
          "path": "/api/v1/job/nj141/data/data/paper.Rmd",
          "name": "paper.Rmd",
          "type": "text/plain",
          "size": 346512
        }
        {
          "path": "/api/v1/job/nj141/data/data/Dockerfile",
          "name": "Dockerfile",
          "type": "text/plain",
          "size": 1729
        }
      ]
    }
  ]
}
```

`path` property

The `path` property for each file in the listing is a link to the raw file. Additionally the `GET` parameter `?size=…` can be appended to retrieve previews of the files. In the case of Images (`png`, `jpg`, `gif`, `tiff`), the value defines the maximum width/height. For text files (`txt`, `csv`, scripts), the value defines the amount of lines returned.

`type` property

The `type` property is a best guess for the MIME type of the file content. It is a result of the files extension. Look at the list of extension to type mapping below.

File extension to MIME type mappings

This list contains the custom mapping of file extensions to MIME types used in the server.

Extension	MIME type
`.R`, `.r`	`script/x-R`

File inspection: RData

`.RData` files are a binary format for usage with R to save any kind of object (data, functions) using an internal serialisation. The format is not suitable for archival or data exchange, but might be included in a compendium out of negligence by or convenience for the author.

Since the file format is binary and not readable by non-R client applications, the API provides the endpoint `/api/v1/inspection` to retrieve a JSON representation of the objects in an RData file.

Values of objects are provided as JSON arrays following the specifications by the R package `jsonlite`.

Simple data types

`GET /api/v1/inspection/<compendium id>?file=simple.Rdata`

```
200 OK

{
  "aChar":[
    "a"
  ],
  "aDouble":[
    2.3
  ],
  "anInteger":[
    1
  ],
  "aString":[
    "The force is great in o2r."
  ]
}
```

Complex data types

Lists are be nested objects, and vectors are JSON arrays (see `jsonlite` docs for details, defaults are used):

`GET /api/v1/inspection/<compendium id>?file=complex.Rdata`

```
200 OK

{
  "characterVector":[
    "one",
    "two",
    "3"
  ],
  "logicalVector":[
    true,
    true,
    false
  ],
  "numericVector":[
    1,
    2,
    -7,
    0.8
  ],
  "orderedList":{
    "name":[
      "Fred"
    ],
    "mynumbers":[
      1,
      2
    ],
    "age":[
      5.3
    ]
  }
}
```

Data frames and matrices are mapped to JSON arrays of complex objects (see `jsonlite` docs for details, defaults are used):

`GET /api/v1/inspection/<compendium id>?file=matrices.Rdata`

```
200 OK

{
  "dataFrame":[
    {
      "ID":1,
      "Passed":true,
      "Colour":"red"
    },
    {
      "ID":2,
      "Passed":true,
      "Colour":"white"
    },
    {
      "ID":3,
      "Passed":true,
      "Colour":"red"
    },
    {
      "ID":4,
      "Passed":false
    }
  ],
  "namedMatrix":[
    [
      1,
      26
    ],
    [
      24,
      68
    ]
  ]
}
```

Path parameters

- `compendium_id` *mandatory* - the compendium identifier to inspect the file from

Query parameters

- `file` *mandatory* - the name of the file to inspect, or a relative path to a file within the compendium
- `objects` *optional* - the name of objects in the file

If selected objects are not loadable from the file, an `errors` property in the response is given for each problematic object:

`GET /api/v1/inspection/<compendium id>?file=simple.RData&objects=bar,anInteger,foo`

```
200 OK

{
  "anInteger":[
    1
  ],
  "errors":[
    "Error: Object 'bar' does not exist in the file simple.RData",
    "Error: Object 'foo' does not exist in the file simple.RData"
  ]
}
```

Errors

```
400 Bad Request

{"error": "Query parameter 'file' missing"}
```

```
400 Bad Request

{"error": "file 'not_available.Rdata' does not exist in compendium kOSMO"}
```

```
400 Bad Request

{"error": "compendium '12345' does not exist"}
```

```
500 Internal Server Error

{"error": "Error loading objects"}
```

Delete compendium

To delete a compendium **candidate**, an HTTP `DELETE` request can be send to the compendium endpoint.

Important

Once a compendium is not a candidate anymore, it **cannot** be deleted via the API.

Required user level

The user deleting a candidate must be the author or have the required user level.

Request

The following request deletes the compendium with the identifier `12345`, including metadata and files.

```
curl -X DELETE https://…/api/v1/compendium/12345 \
    --cookie "connect.sid=<code string here>"
```

Response

The response has an HTTP status of `204` and an empty body for successful deletion.

```
204 OK
```

Error responses for compendium delete

```
401 Unauthorized

{"error":"not authorized"}
```

```
403 Forbidden

{"error":"user level not sufficient to delete compendium"}
```

```
404 Not Found

{"error":"compendium not found"}
```

Upload via API

Upload a research workspace or full compendium as a compressed `.zip` archive with an HTTP `POST` request using `multipart/form-data`.

The upload is only allowed for logged in users. Upon successful extraction of archive and processing of the contents, the `id` for the new compendium is returned.

Required user level and authentication

The user creating a new compendium must have the required user level. Requests must be authenticated with a cookie `connect.sid`, see user authentication.

```
curl -F "compendium=@compendium.zip;type=application/zip" \
    -F content_type=compendium \
    --cookie "connect.sid=<cookie string here>" \
     https://…/api/v1/compendium
```

```
curl -F "compendium=@path/to/workspace.zip;type=application/zip" \
    -F content_type=workspace \
    --cookie "connect.sid=<cookie string here>" \
     https://…/api/v1/compendium
```

```
200 OK

{"id":"a4Ndl"}
```

Important

After successful upload the **candidate process** must be completed for workspaces.

Body parameters for compendium upload

- `compendium` - The archive file
- `content_type` - Form of archive. One of the following:
  - `compendium` - compendium, which is expected to be complete and valid, for *examination* of a compendium
  - `workspace` - formless workspace, for *creation* of a compendium

Warning

If a complete ERC is submitted as a workspace, it may result in an error, or the contained metadata and other files may be overwritten by the creation process.

Error responses for compendium upload

```
400 Bad Request

{"error":"provided content_type not implemented"}
```

```
401 Unauthorized

{"error":"user is not authenticated"}
```

```
401 Unauthorized

{"error":"user level does not allow compendium creation"}
```

```
422 Unprocessable Entity
```

For local testing you can quickly upload some of the example compendia and workspaces from the erc-examples project.

Public share

Load a research compendium by submitting a link to a cloud resource using an HTTP `POST` request using `multipart/form-data`.

Currently, the following repositories are supported:

- Sciebo
- Zenodo
- Zenodo Sandbox

Common

All repositories use the same API endpoint `https://…/api/v1/compendium`, but with different required/optional parameters.

The upload is only allowed for logged in users.

Required user level and authentication

The user creating a new compendium must have the required user level. Requests must be authenticated with a cookie `connect.sid`, see user authentication.

To run the load from the command line, login on the website and open you browser cookies. Find a cookie issued by `o2r.uni-muenster.de` with the name `connect.sid`. Copy the contents of the cookie into the request example below.

Upon successful download from the public share, the `id` for the new compendium is returned.

```
curl -F "content_type=compendium" \
    -F "share_url=https://uni-muenster.sciebo.de/index.php/s/G8vxQ1h50V4HpuA"  \
    --cookie "connect.sid=<code string here>" \
     https://…/api/v1/compendium
```

```
200 OK

{"id":"b9Faz"}
```

Important

After successful load from a public share, the **candidate process** applies.

Sciebo

Sciebo is a cloud storage service at North Rhine-Westphalian universities. Although it builds on ownCloud and the implementation might be able to handle any ownCloud link, only Sciebo's publish shares are supported by this API.

File selection

Depending on the public share contents different processes are triggered:

1. If a file named `bagit.txt` is found, the directory is checked for Bagit validity
2. If a single zip file is found, the file is extracted, if multiple zip files are found, the filename has to be specified, otherwise an error is returned
3. If a single subdirectory is found, the loader uses that subdirectory as the base directory for loading
4. Depending on the value of `content_type` (see below), the public share contents are treated as a complete compendium or as a workspace

Body parameters for creating compendium from public share

- `share_url` - The Sciebo link to the public share (required)
- `content_type` - Form of archive. One of the following (required):
  - `compendium` - complete compendium
  - `workspace` - formless workspace
- `path` - Path to a subdirectory or a zip file in the public share (optional)
  - default is `/`
  - the leading `/` is optional, the loader supports both ways
  - when a directory has multiple zip files, the path can be used to specify which file is used, e.g. `path=/metatainer.zip`

Examples

```
curl -F "content_type=compendium" \
    -F "share_url=https://uni-muenster.sciebo.de/index.php/s/G8vxQ1h50V4HpuA"  \
    -F "path=/metatainer" \
    --cookie "connect.sid=<code string here>" \
     https://…/api/v1/compendium
```

Error responses for creating compendium from public share

```
401 Unauthorized

{"error":"unauthorized: user level does not allow compendium creation"}
```

```
403 Forbidden

{"error":"public share host is not allowed"}
```

```
422 Unprocessable Entity

{"error":"files with unsupported encoding detected: [{'file':'/tmp/o2r/compendium/ejpmi/data/test.txt','en
```

Example data

For testing purposes you can use the following public share, which contains a few ready-to-use compendia:

`https://uni-muenster.sciebo.de/s/G8vxQ1h50V4HpuA

Zenodo

Body parameters for creating a compendium from a Zenodo record

- Identification of the Zenodo record, one of the folloing is required:
    - `share_url` - The link to the zenodo record (optional). May be a link to https://zenodo.org or https://doi.org
    - `doi` - A DOI resolving to the zenodo record (optional)
    - `zenodo_record_id` - The ID of the zenodo record (optional)
- `content_type` - Form of archive. One of the following (required):
    - `compendium` - complete compendium for *inspection*
    - `workspace` - formless workspace for *creation*
- `filename` - Filename of your compendium. For now, only zip-files are supported. (optional)
    - if no `filename` is provided the first zip file is selected
    - multiple files are currently not supported

There must at least one url parameter that resolves to a zenodo record. I.e. one of the following:

Examples

1. Zenodo Record URL (with optional filename parameter)

```
curl -F "content_type=compendium" \
    -F "zenodo_url=https://sandbox.zenodo.org/record/69114"  \
    -F "filename=metatainer.zip" \
    --cookie "connect.sid=<code string here>" \
     https://…/api/v1/compendium
```

1. DOI

```
curl -F "content_type=compendium" \
    -F "doi=10.5072/zenodo.69114"  \
    --cookie "connect.sid=<code string here>" \
     https://…/api/v1/compendium
```

1. Zenodo Record ID

```
curl -F "content_type=compendium" \
    -F "zenodo_record_id=69114"  \
    --cookie "connect.sid=<code string here>" \
     https://…/api/v1/compendium
```

If the Zenodo record id is supplied through the `doi` or `zenodo_record_id` parameter, or if the `share_url` parameter is a `doi.org` URL, a default base URL for the file download is used as selected by the API maintainer. This may be:

- `https://zenodo.org` or
- `https://sandbox.zenodo.org`

Error responses for creating compendium from a Zenodo record

```
401 Unauthorized

{"error":"unauthorized: user level does not allow compendium creation"}
```

```
403 Forbidden

{"error":"host is not allowed"}
```

```
422 Unprocessable Entity

{"error":"public share URL is invalid"}
```

```
422 Unprocessable Entity

{"error":"DOI is invalid"}
```

```
422 Unprocessable Entity

{"error":"files with unsupported encoding detected: [{'file':'/tmp/o2r/compendium/ejpmi/data/test.txt','en
```

Example data

For testing purposes you can use the following public shares. They contain the a compendium with metadata.

- Sciebo: `https://uni-muenster.sciebo.de/index.php/s/G8vxQ1h50V4HpuA`
- Zenodo: `https://sandbox.zenodo.org/record/69114`

Download compendium

Download compendium files as an archive.

Warning

This download feature does *not* provide access to complete and valid compendia, because it does not comprise an update of the packaging, while it does include brokered metadata files. To download a valid compendium, create a shipment with the appropriate recipient.

Supported formats are as follows:

- `zip`
- `tar`
- `tar.gz`

Requests

`GET /api/v1/compendium/$ID.zip`

```
GET /api/v1/compendium/:id.zip
GET /api/v1/compendium/:id.tar
GET /api/v1/compendium/:id.tar.gz
GET /api/v1/compendium/:id.tar?gzip
GET /api/v1/compendium/:id.zip?image=false
```

URL parameters for compendium download

- `:id` - the compendiums id
- `?gzip` - *only for .tar endpoint* - compress tarball with gzip
- `?image=true` or `?image=false` - include tarball of Docker image in the archive, default is `true`

Response

The response is a file attachment. The suggested file name is available in the HTTP header `content-disposition` using the respective file extension for a file named with the compendium identifier (e.g. `wdpV9.zip`, `Uh1o0.tar`, or `LBIt1.tar.gz`).

The `content-type` header also reflects the respective format, which can take the following values:

- `application/zip` for ZIP archive
- `application/x-tar` for TAR archive
- `application/octet-stream` for gzipped TAR

```
200 OK
Content-Type: application/zip
Transfer-Encoding: chunked
Content-Disposition: attachment; filename="$ID.zip"
X-Response-Time: 13.556ms
```

The zip file contains a comment with the original URL.

```
$ unzip -z CXE1c.zip
Archive:  CXE1c.zip
Created by o2r [https://…/api/v1/compendium/CXE1c.zip]
```

Error responses for compendium download

```
404 Not Found

{"error":"no compendium with this id"}
```

```
400 Bad Request

{"error":"no job found for this compendium, run a job before downloading with image"}
```

Compendium metadata

Basics

Metadata in a compendium is stored in a directory `.erc`. This directory contains the normative metadata documents using a file naming scheme `<PREFIX>_<MODEL>_<VERSION>.<FORMAT>` filled via each metadata mapping file found in the broker tool of the o2r metadata tool suite, the default prefix is `metadata`, e.g. `metadata_o2r_1.json`, `metadata_zenodo_1.json`, or `metadata_datacite_41.xml`. The filename of the extracted raw metadata has no versioning and is constantly found as `metadata_raw.json`.

A copy of the files in this directory is kept in database for easier access, so every compendium returned by the API can contain different sub-properties in the metadata property. *This API always returns the database copy of the metadata elements.* You can download the respective files to access the normative metadata documents.

Metadata formats

The files are available on demand, but metadata variants are created after each metadata update.

The sub-properties of the `metadata` and their content are

- `raw` contains raw metadata extracted automatically
- `o2r` holds the **main information for display** and is modelled according the the o2r metadata model. This metadata is reviewed by the user and the basis for translating to other metadata formats and also for search.
- `zenodo` holds Zenodo metadata for shipments made to Zenodo and is brokered from `o2r` metadata
- `zenodo_sandbox` holds Zenodo metadata for shipments made to Zenodo Sandbox, i.e. a clone of `zenodo` metadata

Note

The information in each sub-property are subject to independent workflows and may differ from one another. The term **brokering** is used for translation from one metadata format into another.

Metadata validation

Only valid metadata can be saved to a compendium. The `o2r` metadata element is validated against a JSON Schema using the `validate` tool of `o2r-meta`. The schema file is included in the `o2r-meta` repository: https://raw.githubusercontent.com/o2r-project/o2r-meta/master/schema/json/o2r-meta-schema.json.

Get all compendium metadata

`curl https://…/api/v1/$ID`

`GET /api/v1/compendium/:id`

Abbreviated example response:

```
200 OK

{
  "id":"12345",
  "metadata": {
    "raw": {
      "title": "Programming with Data. Springer, New York, 1998. ISBN 978-0-387-98503-9.",
      "author": "John M. Chambers",
      …
    },
    "o2r": {
      "title": "Programming with Data",
      "creators": [
          {
              "name": "John M. Chambers"
          }
      ],
      "publication_date": 1998,
      …
    },
    "zenodo": {
      …
    }
  },
  "created": …,
  "files": …
}
```

Get o2r metadata

The following endpoint allows to access *only* the normative o2r-metadata element:

`curl https://…/api/v1/$ID/metadata`

`GET /api/v1/compendium/:id/metadata`

```
200 OK

{
  "id":"compendium_id",
  "metadata": {
    "o2r": {
      …
    }
  }
}
```

URL parameters

- `:id` - compendium id

Spatial metadata

For discovery purposes, the metadata includes extracted GeoJSON bounding boxes based on data files in a workspace.

Currently supported spatial data sources:

- shapefiles

The following structure is made available per file:

```
    "spatial": {
        "files": [
    "files": [
        {
            "geojson": {
                "bbox": [
                    -2.362060546875,
                    52.0862573323384,
                    -1.285400390625,
                    52.649729197309426
                ],
                "geometry": {
                    "coordinates": [
                        [
                            [
                                -2.362060546875,
                                52.0862573323384
                            ],
                            [
                                -1.285400390625,
                                52.649729197309426
                            ]
                        ]
                    ],
                    "type": "Polygon"
                },
                "type": "Feature"
            },
            "source_file": "path/to/file1.geojson"
        },
        {
            "geojson": {
                "bbox": [
                    7.595369517803192,
                    51.96245837645124,
                    7.62162297964096,
                    51.96966694957956
                ],
                "geometry": {
                    "coordinates": [
                        [
                            [
                                7.595369517803192,
                                51.96245837645124
                            ],
                            [
                                7.62162297964096,
                                51.96966694957956
                            ]
                        ]
                    ],
                    "type": "Polygon"
                },
                "type": "Feature"
            },
            "source_file": "path/to/file2.shp"
        }
    ],
    "union": {
        "geojson": {
            "bbox": [
                -2.362060546875,
                51.96245837645124,
                7.62162297964096,
                51.96245837645124
            ],
            "geometry": {
                "coordinates": [
                    [
                        -2.362060546875,
                        51.96245837645124
                    ],
                    [
                        7.62162297964096,
                        51.96245837645124
                    ],
                    [
                        7.62162297964096,
                        52.649729197309426
                    ],
                    [
                        -2.362060546875,
                        52.649729197309426
                    ]
                ],
                "type": "Polygon"
            },
            "type": "Feature"
        }
    }
}
```

The `spatial` key has a `union` bounding box, that wraps all extracted bounding boxes.

Update metadata

The following endpoint can be used to update the `o2r` metadata elements. All other metadata sub-properties are only updated by the service itself, i.e. brokered metadata. After creation the metadata is persisted to both files and database, so updating the metadata via this endpoint allows to trigger a brokering process and to retrieve different metadata formats either via this metadata API or via downloading the respective file using the download endpoint.

Metadata update rights

Only authors of a compendium or users with the required user level can update a compendium's metadata.

Metadata update request

```
curl -H 'Content-Type: application/json' \
  -X PUT \
  --cookie "connect.sid=<code string here>" \
  -d '{ "o2r": { "title": "Blue Book" } }' \
  /api/v1/compendium/:id/metadata
```

The request *overwrites* the existing metadata properties, so the *full* o2r metadata must be put with a JSON object called `o2r` at the root, even if only specific fields are changed.

Note

This endpoint allows only to update the `metadata.o2r` elements. All other properties of

URL parameters

- `:id` - compendium id

Metadata update response

The response contains an excerpt of a compendium with only the o2r metadata property.

```
200 OK

{
  "id":"compendium_id",
  "metadata": {
    "o2r": {
      "title": "Blue Book"
    }
  }
}
```

Metadata update error responses

```
401 Unauthorized

{"error":"not authorized"}
```

```
400 Incomplete metadata (description property missing)

{
    "error":"Error updating metadata file, see log for details",
    "log": "[o2rmeta] 20180302.085940 received arguments: {'debug': True, 'tool': 'validate', 'schema': 's
    [o2rmeta] 20180302.085940 launching validator
    [o2rmeta] 20180302.085940 checking metadata_o2r_1.json against o2r-meta-schema.json
    [o2rmeta] 20180302.085940 !invalid: None is not of type 'string'

    Failed validating 'type' in schema['properties']['description']:
        {'type': 'string'}

        On instance['description']:
            None"
}
```

```
400 Bad Request

"SyntaxError [...]"
```

```
422 Unprocessable Entity

{"error":"JSON with root element 'o2r' required"}
```

Other metadata properties

Besides the `metadata` element, a compendium persists some additional properties to reduce computation on the server, and to allows client applications to improve the user experience.

- `bag` - a boolean showing if the uploaded artefact was detected as a BagIt bag (detection file: `bagit.txt`)
- `compendium` - a boolean showing if the uploaded artefact was detected as a compendium (detection file: `erc.yml`)

Example:

(Properties `metadata` and `files` not shown for brevity.)

```
{

    "id": "U9IZ7",
    "metadata": {},
    "created": "2017-01-01T00:00:42.000Z",
    "user": "0000-0002-1825-0097",
    "bag": false,
    "compendium": false,
    "files": {}
}
```

Substitution

Substitution is the combination of an base compendium, "base" for short, and an overlay compendium, or "overlay". A user can choose files from the overlay to replace files of the base, or upload new files. Additionally the user can choose, if the metadata of the base ERC will be adopted for substitution (`keepBase`) or there will be a new extraction of the metadata for the substituted ERC. This new extraction is divided into two choices. The user can let the new extracted metadata be merged into the existing metadata of the base ERC (`extractAndMerge` - **not implemented**) or just save the extracted metadata (`extract` - **not implemented**).

Create substitution

`Create substitution` produces a new compendium with its own files in the storage and metadata in the database. A substitution can be created with an HTTP `POST` request using `multipart/form-data` and content-type `JSON`. Required content of the request are the identifiers of the base and overlay compendia and at least one pair of *substitution files*, consisting of a base file and an overlay file.

Note

A substitution process removes potentially existing packaging information, i.e. if the base compendium was a BagIt bag, the substitution will only contain the payload directory contents (`/data` directory).

The overlay file is stripped of all paths and is copied directly into the substitution's root directory.

Request

`POST /api/v1/substitution`

Request body for a new substitution:

```
{
  "base": "G92NL",
  "overlay": "9fCTR",
  "substitutionFiles": [
    {
      "base": "climate-timeseries.csv",
      "overlay": "mytimeseries_data.csv"
    }
  ],
  "metadataHandling": "keepBase"
}
```

Request body properties

- `base` - id of the base compendium
- `overlay` - id of the overlay compendium
- `substitutionFiles` - array of file substitutions specified by `base` and `overlay`
- `base` - name of the file from the base compendium
- `overlay` - name of the overlay compendium that is exchanged for the original file

- `metadataHandling` - property to specify, if the metadata of the base ERC will be adopted (`keepBase` = **keep metadata** of base ERC) or there will be a new extraction of metadata, that will be merged into the metadata of the base ERC (`extractAndMerge` = **extract and merge metadata** for new ERC) or that will not be merged (`extract` = **extract metadata** of new ERC)

Required user level

The user creating a new substitution must have the required user level.

Response

```
201 CREATED

{
  "id": "oMMFn"
}
```

Error responses

```
401 Unauthorized

{"error":"not authenticated"}
```

```
401 Unauthorized

{"error":"not allowed"}
```

```
404 Not Found

{"error":"base compendium not found"}
```

```
404 Not Found

{"error":"overlay compendium not found"}
```

View substituted Compendium

Request

`curl https://.../api/v1/compendium/$ID`

`GET /api/v1/compendium/:id`

This request is handled as regular GET request of a compendium (see View single compendium).

Response

A substituted compendium is be saved as a usual compendium, but with additional metadata specifying this as a substituted compendium and giving information about the substitution.

Example 01 - in case there are no conflicts between filenames of any base file and overlay file :

```
200 OK

{
  "id": "oMMFn",
  ...
  "metadata": {
      ...
      "substitution": {
          "base": "G92NL",
          "overlay": "9fCTR",
          "substitutionFiles": [
            {
              "base": "climate-timeseries.csv",
              "overlay": "mytimeseries_data.csv",
              "filename": "climate-timeseries.csv"
            }
          ],
          "metadataHandling": "keepBase"
      },
      ...
  },
  "substituted": true,
  ...
}
```

Example 02 - in case the overlay file has the same filename as one of the existing base files and is in a sub-directory in the overlay compendium:

```
200 OK

{
  "id": "oMMFn",
  ...
  "metadata": {
      ...
      "substitution": {
          "base": "G92NL",
          "overlay": "9fCTR",
          "substitutionFiles": [
            {
              "base": "climate-timeseries.csv",
              "overlay": "dataFiles/input.csv",
              "filename": "overlay_input.csv"
            }
          ],
          "metadataHandling": "keepBase"
      },
      ...
  },
  "substituted": true,
  ...
}
```

Response additional metadata

- `metadata.substitution` - object, specifying information about the substitution
    - `base` - id of the base compendium
    - `overlay` - id of the overlay compendium
    - `substitutionFiles` - array of file substitutions specified by `base` and `overlay`
        - `base` - name of the file from the base ERC
        - `overlay` - name of the file from the overlay ERC
        - `filename` - as seen in the examples above, `filename` will be created. If there is a conflict with any basefilename and an overlayfilename, the overlayfilename will get an additional "**overlay_**" prepended (see Example 02). *(optional add)*
    - `metadataHandling` - property to specify, if the metadata of the base ERC will be adopted (`keepBase` = **keep metadata** of base ERC) or there will be a new extraction of metadata, that will be merged into the metadata of the base ERC (`extractAndMerge` = **extract and merge metadata** for new ERC) or that will not be merged (`extract` = **extract metadata** of new ERC)
- `substituted` - will be set `true`

List substituted Compendia

Request

`curl https://.../api/v1/substitution`

`GET /api/v1/substitution`

Response

The result is a list of compendia ids which were created by a substitution process.

```
200 OK

{
  "results": [
    "oMMFn",
    "asdi5",
    "nb2sg",
    …
  ]
}
```

If there are no substitutions yet, the returned list is empty.

```
200 OK
{
  "results": [ ]
}
```

Filter results with following parameters:

`curl https://.../api/v1/substitution?base=$BASE_ID&overlay=$OVERLAY_ID`

`GET /api/v1/substitution?base=base_id&overlay=overlay_id`

- Filter by `base`:

`curl https://.../api/v1/substitution?base=jfL3w`

`GET /api/v1/substitution?base=jfL3w`

Result is a list of substituted compendia based on the given base compendium:

```
200 OK

{
  "results": [
    "wGmFn",
    …
  ]
}
```

- Filter by `overlay`:

`curl https://.../api/v1/substitution?overlay=as4Kj`

`GET /api/v1/substitution?overlay=as4Kj`

Result is a list of substituted compendia based on the given overlay compendium:

```
200 OK

{
  "results": [
    "9pQ34",
    "1Tnd3",
    …
  ]
}
```

- Filter by `base` and `overlay`:

`curl https://.../api/v1/substitution?base=lO3Td&overlay=as4Kj`

`GET /api/v1/substitution?base=lO3Td&overlay=as4Kj`

Result is a list of substituted compendia based on the given base *and* overlay compendium:

```
200 OK

{
  "results": [
    "9pQ34",
    …
  ]
}
```

Error responses

```
401 Unauthorized

{"error":"not authenticated"}
```

```
401 Unauthorized

{"error":"not allowed"}
```

```
404 Not Found

{"error":"base compendium not found"}
```

```
404 Not Found

{"error":"overlay compendium not found"}
```

```
400 Not Found

{"error":"base file is undefined"}
```

```
400 Not Found

{"error":"overlay file is undefined"}
```

URL parameters for substituted compendium lists

- `:base` - id of the base compendium that the results should be related to
- `:overlay` - id of the overlay compendium that the results should be related to

Execute a compendium

Execution jobs are used to run the analysis in a compendium. When a new execution job is started, the contents of the research compendium are cloned to create a trackable execution (see Job files). The status information, logs and final working directory data are saved in their final state, so that they can be reviewed later on.

All execution jobs are tied to a single research compendium and reflect the execution history of that research compendium.

A trivial execution job would be a completely unmodified compendium, to test the executability and thus basic reproducibility of the contained workflow.

Job files

All files except the following are copied to a separate storage for each job:

- existing image tarballs, e.g. `image.tar` to reduce size of copied files and because jobs use images from the local image repository anyway
- the *display file* to make sure the check does not wrongly work on the original display file

Job status

The property `job.status` shows the **overall status** of a job.

The overall status can be one of following:

- `success` - if status of all steps is `success`.
- `failure` - if status of at least one step is `failure`.
- `running` - if status of at least one step is `running` and no status is `failure`.

More information about `steps` can be found in subsection `Steps` of section `View single job`.

Steps of a job

One job consists of a series of steps. The are executed in order.

- **validate_bag** Validate the BagIt bag using the npm library bagit; may be skipped if compendium is not a bag, will usually fail because of added metadata files during upload.
- **generate_configuration** Create a compendium configuration file; may be skipped if configuration file is already present.
- **validate_compendium** Parses and validates the bagtainer configuration file.
- **generate_manifest** Executes the given analysis to create a container manifest; may be skipped if manifest file is already present.

- **image_prepare** Create an archive of the payload (i.e. the workspace, or the data in a BagIt bag), which allows to build and run the image also on remote hosts.
- **image_build** Build an image and tag it `erc:<job_id>`.
- **image_execute** Run the container and return based on status code of program that ran inside the container.
- **check** Run a check on the contents of the container. Validate the results of the executed calculations. The check provides either a list of errors or a reference to displayable content in the property `display.diff`.
- **image_save** Export the image to a file within the compendium directory (potentially a large file!). This is skipped if the check failed.
- **cleanup** Remove image or job files (depending on server-side settings).

Status

All steps can have one of the following status:

- `queued`: step is not yet started
- `running`: step is currently running
- `success`: step is completed successfully - positive result
- `failure`: step is completed unsuccessfully - negative result
- `skipped`: step does not fit the given input or results of previous steps, e.g. bag validation is not done for non-bag workspaces - neutral result

Step metadata

Additional explanations on steps' status will be transmitted in the `text` property. `text` is an array, with the latest element holding the newest information. During long running steps, the `text` element is updated by appending new information when available.

The `start` and `end` timestamps indicate the start and end time of the step. They are formatted as ISO8601.

Specific steps may carry more information in additional properties.

New job

Create and run a new execution job with an HTTP `POST` request using `multipart/form-data`. Requires a `compendium_id`.

Required user level and authentication

The user creating a new compendium must have the required user level. Requests must be authenticated with a cookie `connect.sid`, see user authentication.

`curl -F compendium_id=$ID https://…/api/v1/job`

`POST /api/v1/job`

```
200 OK

{"job_id":"ngK4m"}
```

Body parameters for new jobs

- `compendium_id` (`string`): **Required** The identifier of the compendium to base this job on.

Error responses for new jobs

```
404 Not Found

{"error":"no compendium with this ID found"}
```

```
500 Internal Server Error

{"error":"could not create job"}
```

List jobs

Lists jobs with filtering and pagination, returning up to 100 results by default.

Results are be sorted by descending date of last change. The content of the response can be limited to certain properties of each result by providing a list of fields, i.e. the parameter `fields`.

Results can be filtered:

- by `compendium_id` i.e. `compendium_id=a4Dnm`,
- by `status` i.e. `status=success` or
- by `user` i.e. `user=0000-0000-0000-0001`

`curl https://…/api/v1/job?limit=100&start=2&compendium_id=$ID&status=success&fields=status`

`GET /api/v1/job?limit=100&start=2&compendium_id=a4Dnm&status=success`

```
200 OK

{
  "results": [
    "nkm4L",
    "asdi5",
    "nb2sg",
    …
  ]
}
```

The overall job state can be added to the job list response:

`GET /api/v1/job?limit=100&start=2&compendium_id=a4Dnm&status=success&fields=status`

```
200 OK

{
  "results": [
    {
      "id":"nkm4L",
      "status":"failure"
    },
    {
      "id":"asdi5",
      "status":"success"
    },
    {
      "id":"nb2sg",
      "status":"running"
    },
    …
  ]
}
```

If there are no jobs, the returned list is empty:

```
200 OK
{
  "results": [ ]
}
```

GET query parameters for listing jobs

- `compendium_id` - Comma-separated list of related compendium ids to filter by.
- `start` - Starting point of the result list. `start - 1` results are skipped. Defaults to 1.
- `limit` - Limits the number of results in the response. Defaults to 100.
- `status` - Specify status to filter by. Can contain following `status`: `success`, `failure`, `running`.
- `user` - Public user identifier to filter by.
- `fields` - Specify which additional attributes results list should contain. Can contain following fields: `status`, `user`. Defaults to none.

View single job

View details for a single job. The file listing format is described in compendium files.

`curl https://…/api/v1/job/$ID?steps=all`

`GET /api/v1/job/:id?steps=all`

```
200 OK

{
  "id":"UMmJ7",
  "compendium_id":"BSgxj",
  "steps":{
    "validate_bag":{
      "status":"skipped",
      "text":[
        "Not a bag"
      ],
      "end":"2017-11-17T13:22:48.105Z",
      "start":"2017-11-17T13:22:48.105Z"
    },
    "generate_configuration":{
      "status":"success",
      "text":[
        "configuration file not found, generating it...",
        "Saved configuration file to job and compendium"
      ],
      "end":"2017-11-17T13:22:48.119Z",
      "start":"2017-11-17T13:22:48.113Z"
    },
    "validate_compendium":{
      "status":"success",
      "text":[
        "all checks passed"
      ],
      "end":"2017-11-17T13:22:48.127Z",
      "start":"2017-11-17T13:22:48.125Z"
    },
    "generate_manifest":{
      "status":"success",
      "text":[
        /* abbreviated */
        "INFO [2017-11-17 13:22:56] Going online? TRUE  ... to retrieve system dependencies (sysreq-api)",
        "INFO [2017-11-17 13:22:56] Trying to determine system requirements for the package(s) 'knitr, bac
        "INFO [2017-11-17 13:22:58] Adding CRAN packages: backports, digest, evaluate, htmltools, knitr, m
        "INFO [2017-11-17 13:22:58] Created Dockerfile-Object based on /erc/main.Rmd",
        "INFO [2017-11-17 13:22:58] Writing dockerfile to /erc/Dockerfile",
        /* abbreviated */
        "generated manifest"
      ],
      "manifest":"Dockerfile",
      "end":"2017-11-17T13:22:58.865Z",
      "start":"2017-11-17T13:22:48.129Z"
    },
    "image_prepare":{
      "status":"success",
      "text":[
        "payload with 756224 total bytes created"
      ],
      "end":"2017-11-17T13:22:58.906Z",
      "start":"2017-11-17T13:22:58.875Z"
    },
    "image_build":{
      "status":"success",
      "text":[
        "Step 1/6 : FROM rocker/r-ver:3.4.2",
        "---> 3cf05960bf30",
        /* abbreviated */
        "---> Running in eb7ccd432592",
        "---> 84db129215f6",
        "Removing intermediate container eb7ccd432592",
        "Successfully built 84db129215f6",
        "Successfully tagged erc:UMmJ7"
      ],
      "end":"2017-11-17T13:22:59.899Z",
      "start":"2017-11-17T13:22:58.912Z"
    },
    "image_execute":{
      "status":"success",
      "text":[
        "[started image execution]",
        /* abbreviated */
        "Output created: display.html\r\n> \r\n>",
        "[finished image execution]"
      ],
      "statuscode":0,
      "start":"2017-11-17T13:22:59.904Z"
    },
    "check":{
      "status":"failure",
      "text":[
        "Check failed"
      ],
      "images":[
        {
          "imageIndex":0,
          "resizeOperationCode":0,
          "compareResults":{
            "differences":204786,
            "dimension":1290240
          }
        }
      ],
      "display":{
        "diff":"/api/v1/job/UMmJ7/data/check.html"
      },
      "errors":[ ],
      "checkSuccessful":false,
      "end":"2017-11-17T13:23:04.439Z",
      "start":"2017-11-17T13:23:03.479Z"
    },
    "image_save": {
      "status": "success",
      "text": [
        "[Saving image tarball file]",
        "[Saved image tarball to file (size: 875.14 MB)]"
      ],
      "start": "2018-01-29T17:38:55.111Z",
      "file": "image.tar",
      "end": "2018-01-29T17:39:36.845Z"
    },
    "cleanup":{
      "status":"success",
      "text":[
        "Running regular cleanup",
        "Removed image with tag erc:UMmJ7: [{\"Untagged\":\"erc:UMmJ7\"},{\"Deleted\":\"sha256:84db129215f
        "Deleted temporary payload file."
      ],
      "end":"2017-11-17T13:23:05.592Z",
      "start":"2017-11-17T13:23:04.575Z"
    }
  },
  "status":"failure",
  "files":{ /* see compendium */  }
}
```

URL parameters for single job view

- `:id` - id of the job to be viewed
- `steps` - Steps to be loaded with full details

The properties `status`, `start` and `end` of *all steps* are always included in the response.

Supported values for `steps` are `all` or a comma separated list of one or more step names, e.g. `generate_configuration,check`. The response will contain the default properties for all steps but other properties only for the selected ones. Any other values for `steps` or not providing the parameter at all will return the default (e.g. `steps=no`).

Error responses for single job view

```
404 Not Found

{"error":"no compendium with this ID found"}
```

Job status updates

You can subscribe to real time status updates on jobs using WebSockets. The implementation is based on socket.io and using their client is recommended.

The job log is available at `https://o2r.uni-muenster.de` under the namespace `api/v1/logs/job`.

```
# create a socket.io client:
var socket = io('https://o2r.uni-muenster.de/api/v1/logs/job');
```

Search

The search uses a document database to provide high speed and powerful search capabilities for compendia, including spatial and temporal properties.

The search structure is based on Elasticsearch and thereby eases an implementation, because the requests and responses shown here can be directly mapped to respectively from Elasticsearch's API.

Indexed information:

- compendium *metadata* (including harvested and user-edited metadata such as temporal ranges and spatial extents)
- *full texts* of text files in a compendium

Simple search

A simple search allows searching for search terms using an `HTTP GET` request accepting `application/json` content type.

`curl -H 'Content-Type: application/json' https://.../api/v1/search?q=$SEARCHTERM`

`GET /api/v1/search?q=Reproducible`

`GET /api/v1/search?q=great reproducible research`

The **response** is `JSON` with the root element is `hits`, which has the same as the `hits` element from an Elasticsearch response but does not include internal fields such as `_index`, `_type`, and `_id`.

```
200 ok

{
  "hits": {
    "total": 1,
    "max_score": 1.0586987,
    "hits": [
      {
        "_score": 1.0586987,
        "_source": {
          "metadata": {
            "o2r": ...
          },
        }
      }
    ]
  }
}
```

Note

The available metadata is a synced clone of the compendium metadata stored in the main database. For more information on the mapping from the main database to the search database, take a look at the `o2r-finder` microservice.

Query parameters for simple search

- `q` - search term(s), must be URL-encoded
- `resources` - a comma-separated list of resources to include in the search; supported values are
- `compendia`
- `jobs`
- `all` (default)

Example requests

- http://o2r.uni-muenster.de/api/v1/search?q=*
- http://o2r.uni-muenster.de/api/v1/search?q=europe temperature data analysis
- http://o2r.uni-muenster.de/api/v1/search?q=europe%20temperature%20data%20analysis
- http://o2r.uni-muenster.de/api/v1/search?q=10.5555%2F12345678
- http://o2r.uni-muenster.de/api/v1/search?q=geo&resources=compendia
- http://o2r.uni-muenster.de/api/v1/search?q=failure&resources=jobs

Complex Search

A complex search is enabled via `POST` requests with a `JSON` payload as `HTTP POST` data (*not* `multipart/form-data`) accepting an `application/json` content type as response. Queries can include filters, aggregation and spatio-temporal operations as defined in the Elasticsearch Query DSL.

```
curl -X POST -H 'Content-Type: application/json' 'https://.../api/v1/search' -d '$QUERY_DSL'
```

The **response** structure is the same as for simple search.

Note

Use the index names `compendia` and `jobs` in a terms query to only retrieve one resource type.

Query fields for complex search

The following fields are especially relevant to build queries.

- `metadata.o2r.temporal.begin` and `metadata.o2r.temporal.end` provide a compendium's temporal extent
- `metadata.o2r.spatial.geometry` has the compendium's spatial extent

Besides these fields, all metadata of the `o2r` metadata format can be used.

Examples

Temporal search

```
POST /api/v1/search -d '{
  "query": {
      "bool": {
          "must": {
              "match_all": {}
          },
          "filter": [
              {
                  "range": {
                      "metadata.o2r.temporal.begin": {
                          "from": "2015-03-01T00:00:00.000Z"
                      }
                  }
              },
              {
                  "range": {
                      "metadata.o2r.temporal.end": {
                          "to": "2017-10-01T00:00:00.000Z"
                      }
                  }
              }
          ]
      }
  },
  "from": 0,
  "size": 10
}'
```

Spatial search

```
{
    "bool": {
        "must": {
            "match_all": {}
         },
         "filter": {
              "geo_shape": {
                   "metadata.o2r.spatial.geometry": {
                        "shape": {
                            "type": "polygon",
                            "coordinates": [... GeoJSON coordinates...]
                         },
                         "relation": "within"
                    }
               }
          }
     }
}
```

In this example a filter has been nested within a boolean/must match query. The filter has been applied to the `metadata.o2r.spatial.geometry` field of the dataset with a `within` relation so that only compendia with a spatial extent completely contained in the provided shape are fetched.

Resource search

```
{
    "query": {
        "terms": {
            "_index": ["compendia"]
        }
    }
}
```

Response

```
200 ok

{
  "hits": {
    "total": 1,
    "max_score": 1.0586987,
    "hits": [
      {
        "_score": 1.0586987,
        "_source": {
          "metadata": {
            "o2r": ...
          },
        }
      }
    ]
  }
}
```

Ship compendia and metadata

Shipments are used to deliver compendia and their metadata to third party repositories or archives. This section covers shipment related requests, including repository file management.

Packaging

The packaging of a compendium ensures a recipient can verify the integrity of the transported data. Currently, the shipment process always creates BagIt bags to package a compendium.

Supported recipients

Use the *recipient* endpoint to find out, which repositories are available and configured. The response is list of tuples with `id` and `label` of each repository. The `id` is the repository identifier to be used in requests to the `/shipment` endpoint, e.g. to define the recipient, while `label` is a human-readable text string suitable for display in user interfaces.

`GET /api/v1/recipient`

```
200

{
    "recipients": [{
        "id": "download",
        "label": "Download"
    }, {
        "id": "b2share_sandbox",
        "label": "Eudat b2share Sandbox"
    }, {
        "id": "zenodo_sandbox",
        "label": "Zenodo Sandbox"
    }]
}
```

An implementation may support one or more of the following repositories:

- `b2share` - Eudat b2share
- `b2share_sandbox` - Eudat b2share Sandbox
- `zenodo` - Zenodo Sandbox
- `zenodo_sandbox` - Zenodo Sandbox

The `download` recipient is a surrogate to enable shipping to the user's local storage.

List shipments

This is a basic request to list all shipment identifiers.

`GET /api/v1/shipment`

```
200
["dc351fc6-314f-4947-a235-734ab5971eff", "..."]
```

You can also get only the shipment identifiers belonging to a compendium id (e.g. `4XgD97`).

`GET /api/v1/shipment?compendium_id=4XgD97`

URL parameter:

- `compendium_id` - the identifier of a specific compendium

```
200
["dc351fc6-314f-4947-a235-734ab5971eff", "..."]
```

Get a single shipment

`GET /api/v1/shipment/dc351fc6-314f-4947-a235-734ab5971eff`

```
200
{
  "last_modified": "2016-12-12 10:34:32.001475",
  "recipient": "zenodo",
  "id": "dc351fc6-314f-4947-a235-734ab5971eff",
  "deposition_id": "63179",
  "user": "0000-0002-1825-0097",
  "status": "shipped",
  "compendium_id": "4XgD97",
  "deposition_url": "https://zenodo.org/record/63179"
}
```

Note

Returned deposition URLs (property `deposition_url`) from Zenodo as well as Eudat b2share (records) will only be functional after publishing.

Create a new shipment

You can start a initial creation of a shipment, leading to transmission to a repository and creation of a deposition, using a `POST` request.

`POST /api/v1/shipment`

This **requires** the following parameters as `multipart/form-data` or `application/x-www-form-urlencoded` encoded data:

- `compendium_id` (`string`): the id of the compendium
- `recipient` (`string`): identifier for the repository

The following are **optional parameters**:

- `update_packaging` (`boolean`, default: `false`): the shipment creation only succeeds if a valid package is already present under the provided compendium identifier, or if no packaging is present at all and a new package can be created. In case a partial or invalid package is given, this parameter can control the shipment creation process: If it is set to `true`, the shipment package is updated during the shipment creation in order to make it valid, if set to `false` the shipment creation results in an error.

- `cookie` (`string`): an authentication cookie must be set in the request header, but it may also be provided via a `cookie` form parameter as a fallback
- `shipment_id` (`string`): a user-defined identifier for the shipment (see `id` in response)

Required user level

The user sending the request to create a shipment must have the required user level.

Creation response

The response contains the shipment document, see Get a single shipment. Some of the fields are not available (have value `null`) until after publishing, e.g. `deposition_url`.

```
201

{
  "id": "9ff3d75e-23dc-423e-a6c6-6987ac5ffc3e",
  "recipient": "zenodo",
  "status": "shipped",
  "deposition_id": "79102"
}
```

If the recipient is the **download** surrogate, the response will be `202` and a zip stream with the Content type `application/zip`.

```
202
```

(zip stream starting point)

The download zip stream is also available under the url of the shipment plus `/dl`, once it has been created, e.g.:

```
http://localhost:8087/api/v1/shipment/22e7b17c-0047-4cb9-9041-bb87f30de388/dl
```

Shipment status

A shipment can have three possible status:

- `shipped` - a deposition has been created at a repository and completed the necessary metadata for publication.
- `published` - the contents of the shipment are published on the repository, in which case the publishment can not be undone.
- `error` - an error occurred during shipment or publishing.

To get only a shipment's current status you may use the sub-resource `/status`:

`GET api/v1/shipment/<shipment_id>/status`

```
200

{
"id": "9ff3d75e-23dc-423e-a6c6-6987ac5ffc3e",
"status": "shipped"
}
```

Publish a deposition

The publishment is supposed to have completed the status `shipped` where metadata requirements for publication have been checked.

Note

Once published, a deposition can no longer be deleted on the supported repositories.

`PUT api/v1/shipment/<shipment_id>/publishment`

```
200

{
"id": "9ff3d75e-23dc-423e-a6c6-6987ac5ffc3e",
"status": "published"
}
```

Note that a publishment is not possible if the recipient is the download surrogate which immediately results in a zip stream as a response.

Files in a deposition

List deposition files

You can request a list of all files in a deposition and their properties with the sub-resource `/publishment`.

`GET api/v1/shipment/<shipment_id>/publishment`

```
200

{
"files": [{
    "filesize": 393320,
    "id": "bae2a60c-bd59-47e1-a443-b34bb7d0a981",
    "filename": "4XgD9.zip",
    "checksum": "702f4db3e53b22176d1d5ddcda462a27",
    "links": {
        "self": "https://sandbox.zenodo.org/api/deposit/depositions/71552/files/bae2a60c-bd59-47e1-a443-b3
        "download": "https://sandbox.zenodo.org/api/files/31dc8f3d-df00-4d8a-bd99-64ef341372b3/4XgD9.zip"
    }
}]
}
```

You can find the `id` of the file you want to interact with in this json list object at `files[n].id`, where `n` is the position of that file in the array. Files can be identified in this response by either their id in the depot, their filename or their checksum.

Delete a specific file from a deposition

You can delete files from a `shipped` shipment's deposition. You must state a file's identifier, which can be retrieved from the shipment's deposition files property `id`, as the `file_id` path parameter. Files for a `published` shipment usually cannot be deleted.

`DELETE api/v1/shipment/<shipment_id>/files/<file_id>`

```
204
```

Error responses

```
400

{"error":"bad request"}
```

```
403

{"error": "insufficient permissions"}
```

User

List users

Return a list of user ids. Pagination (including defaults) as described for compendia is available for users.

`curl https://…/api/v1/user`

`GET /api/v1/user`

```
200 OK

{
    "results": [
        "0000-0002-1825-0097",
        "0000-0002-1825-0097"
    ]
}
```

If there are no users, the returned list is empty:

```
200 OK
{
  "results": [ ]
}
```

Pagination is supported using the query parameters `start` and `limit`.

- `limit` is the number of results in the response, defaults to `10`. It numeric and larger than `0`.
- `start` is the index of the first list item in the response, defaults to `1`. It must be numeric and larger than `0`.

`GET /api/v1/user?start=5&limit=10`

Error responses for user list

```
400 Bad Request

{"error":"limit must be larger than 0"}
```

View single user

Show the details of a user.

`curl https://…/api/v1/user/$ID`

`GET /api/v1/user/:id`

```
200 OK

{
    "id": "0000-0002-1825-0097",
    "name": "o2r"
}
```

The content of the response depends on the state and level of the user that requests the resource. The above response only contains the id and the publicly visible name. The following response contains more details and requires a certain user level of the authenticated user making the request:

`curl --cookie "connect.sid=<session cookie here>" https://…/api/v1/user/0000-0002-1825-0097`

```
200 OK

{
    "id": "0000-0002-1825-0097",
    "name": "o2r",
    "level": 0,
    "lastseen": "2016-08-15T12:32:23.972Z"
}
```

URL parameters for single user view

- `:id` - the user id

Error responses for single user view

```
404 Not Found

{"error":"no user with this id"}
```

Authentication

User authentication is done via authenticated sessions, which are referenced with a cookie called `connect.sid`. For every endpoint that needs user authentication, a cookie with an authenticated session is required.

Client authentication

To execute restricted operations of the API, such as compendium upload or job execution, a client must provide an authentication token via a cookie.

A client must first login on the website to access a browser cookie issued by `o2r.uni-muenster.de` with the name `connect.sid`. Provide the content of the cookie when making requests to the API as shown in the request example below.

Access authentication information for direct API access

To run commands which require authentication from the command line, a user must login on the website first. Then open you browser cookies and find a cookie issued by `o2r.uni-muenster.de` with the name `connect.sid`. Use the the contents of the cookie for your requests, for example as shown below when using curl.

```
curl [...] --cookie "connect.sid=<code string here>" \
     https://…/api/v1/endpoint
```

Authentication within microservices

Attention: The authentication process *requires* a secured connection, i.e. `HTTPS`.

Authentication provider

Session authentication is done using the OAuth 2.0 protocol. Currently ORCID is the only available authentication provider, therefore users need to be registered with ORCID. Because of its nature, the authentication workflow is not a RESTful service. Users must follow the redirection to the login endpoint with their web browser and grant access to the o2r reproducibility service for their ORCID account. They are then sent back to our authentication service, which verifies the authentication request and enriches the user session with the verified ORCID for this user.

Start OAuth login

Navigate the web browser (e.g. via a HTML `<a>` link) to `/api/v1/auth/login`, which then redirects the user and request access to your ORCID profile. After granting access, ORCID redirects the user back to the `/api/v1/auth/login` endpoint with a unique `code` param that is used to verify the request.

If the verification was successful, the endpoint returns a session cookie named `connect.sid`, which is tied to a authenticated session. The server answers with a `301 redirect`, which redirects the user back to `/`, the start page of the o2r website.

If the login is unsuccessful, the user is not redirected back to the site and no further redirects are configured.

Request authentication status

As the cookie is present in both authenticated and unauthenticated sessions, clients (e.g. web browser user interfaces) must know if their session is authenticated, and if so, as which ORCID user. For this, send a `GET` request to the `/api/v1/auth/whoami` endpoint, including your session cookie.

`curl https://…/api/v1/auth/whoami --cookie "connect.sid=…`

`GET /api/v1/auth/whoami`

```
200 OK

{
  "orcid": "0000-0002-1825-0097",
  "name": "o2r"
}
```

Error response for requests requiring authentication

When no session cookie was included, or the included session cookie does not belong to a authenticated session, the service responds with a `401 Unauthorized` message.

```
401 Unauthorized

{
  "error": "not authenticated"
}
```

User levels

Users are authenticated via OAuth and the actions on the website are limited by the `level` associated with an account. On registration, each account is assigned a level `0`. Only admin users and the user herself can read the level of a user.

The following is a list of actions and the corresponding required *minimum* user level.

- `0` *Users* (everybody)
  - Create new jobs
  - View compendia, jobs, user details
- `100` *Known users*
  - Create new compendium
  - Create shipments
  - Create substitutions
  - Delete own candidates
- `500` *Editors*
  - Edit user levels
  - Edit metadata of other user's compendia
  - View other user's candidates
- `1000` *Admins*
  - Delete candidates
  - View status pages of microservices

Edit user

You can update information of an existing user using the `HTTP` operation `PATCH`.

Change user level request

The user level can be changed with an `HTTP` `PATCH` request. The new level is passed to the API via a query parameter, i.e. `..?level=<new level value>`. The value must be an `int` (integer). The response is the full user document with the updated value.

Required user level

The user sending the request to change the level must have the required user level.

```
curl --request PATCH --cookie "connect.sid=<session cookie here>" \
  https://…/api/v1/user/0000-0002-1825-0097?level=42`
```

```
200 OK

{
    "id": "0000-0002-1825-0097",
    "name": "o2r",
    "level": 42,
    "lastseen": "2016-08-15T12:32:23.972Z"
}
```

Error responses for user level change

```
401 Unauthorized

{
  "error": "user is not authenticated"
}
```

```
401 Unauthorized

{
  "error": "user level does not allow edit"
}
```

```
400 Bad Request

{
  "error": "parameter 'level' could not be parsed as an integer"
}
```

Previous