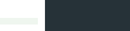

Open Geospatial Consortium

Submission Date 2024-11-21

Approval Date: <yyyy-mm-dd>

Publication Date: <yyyy-mm-dd>

External identifier of this OGC® document: <http://www.opengis.net/doc/[{doc-type/}]{standard}/{m.n}>

Internal reference number of this OGC® document: 24-059

Version: 1.0

Category: OGC® Community Standard

Editor: Matthias Mohr

openEO API Community Standard

Copyright notice

Copyright © 2025 Open Geospatial Consortium

License notice

Apache 2.0

Warning

This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC® Community Standard
Document subtype:
Document stage: Draft
Document language: English

OGC openEO API Community Standard (1.2.0)

Download OpenAPI specification: Download

openEO Project Steering Committee: openeo.psc@uni-muenster.de | URL: https://openeo.org
License: Apache 2.0

openEO Documentation

Introduction

The OGC openEO Community Standard defines an API in support of interoperable cloud-based processing of large Earth observation datasets.

We recommend reading the glossary before diving into this specification. The glossary explains the most important terms used in this specification.

The OGC openEO Community Standard consists of two parts:

- OGC openEO API Community Standard, this API specification
- OGC openEO Processes Community Standard, a set of well-defined processes that is recommended to be implemented by back-end providers and is offered through their openEO API

Abstract

openEO specifies an open application programming interface (API) for connecting applications and other client software to big Earth observation cloud back-ends in a simple and unified way.

The openEO specification aims at increasing the interoperability of big EO data processing of satellite imagery in the cloud. Implementations of openEO can be used to add an interoperability layer on top of existing services. Its development has been driven by the need to overcome the challenges associated with different tools, APIs, and data formats in geospatial technology. openEO has been developed from the bottom up, with each version of the specification supported by implementations.

The primary use case for specifying openEO was to simplify and unify the data processing using a common API and a specification for a set of pre-defined processes. As such, users can still work in their favored programming language without worrying about data organization and pre-processing. Users can avoid vendor lock-in as the generated process descriptions can be executed at multiple provider endpoints, making it easier to compare and reproduce processing results between different providers.

Source of this Document

The majority of the content in this OGC document is a direct copy of the content contained at https://github.com/Open-EO/openeo-api. No normative changes have been made to the content. This OGC document does contain content not in source openEO API GitHub repository. Specifically, while derived from content on the openEO API repository, the chapters "Abstract", "Source of this Document", "Submitting Organizations", and "Supporting Organizations" in this document are not found on the openEO API repository.

Submitting Organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- openEO Project Steering Committee

The organizations listed above have granted the Open Geospatial Consortium (OGC) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version under a Apache License, Version 2.0 (see below).

Supporting Organizations

The following organization (in alphabetical order) support the submission of the openEO Community Standard to the OGC:

- EOX IT Services GmbH
- EUMETSAT
- Eurac Research
- European Space Agency (ESA)
- GeoConnections - Natural Resources Canada
- German Aerospace Center - DLR
- Matthias Mohr - Softwareentwicklung
- Planet Labs PBC
- Telespazio VEGA UK Ltd
- University of Münster - Institute for Geoinformatics
- VITO (Flemish Institute for Technological Research)

License Agreement

The standard is licensed under the Apache License, Version 2.0. You can implement this standard in services, clients or processing tools without restrictions.

Conformance

Conformance class: `https://api.openeo.org/1.2.0`

API Principles

Language

In the specification the key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 and RFC 8174.

Literal Case

Unless otherwise stated the API works **case sensitive**.

All names SHOULD be written in snake case, i.e. words are separated with one underscore character (`_`) and no spaces, with all letters lower-cased. Example: `hello_world`. This applies particularly to endpoints and JSON property names. HTTP header fields are generally case-insensitive according to RFC 7230 and their respective casing conventions are followed, e.g. `Content-Type` or `OpenEO-Costs`, for better readability and consistency.

HTTP / REST

This specification uses HTTP REST Level 2 for communication between client and back-end server.

Public API implementations MUST be available via HTTPS only.

Endpoints are made use meaningful HTTP verbs (e.g. GET, POST, PUT, PATCH, DELETE) whenever technically possible. If there is a need to transfer big chunks of data for a GET requests to the back-end, POST requests MAY be used as a replacement as they support to send data via request body. Unless otherwise stated, PATCH requests are only defined to work on direct (first level) children of the full JSON object. Therefore, changing a property on a deeper level of the full JSON object always requires sending the whole JSON object defined by the first-level property.

Naming rules of the API endpoints follow the REST principles. Therefore, endpoints are centered around resources. Resource identifiers MUST be named with a noun in plural form except for single actions that can not be modelled with the regular HTTP verbs. Single actions MUST be single endpoints with a single HTTP verb (POST is RECOMMENDED) and no other endpoints beneath it.

The openEO API makes use of HTTP Content Negotiation, including, but not limited to, the request headers `Accept`, `Accept-Charset` and `Accept-Language`.

JSON

The API uses JSON for request and response bodies whenever feasible. Services use JSON as the default encoding. Other encodings can be requested using HTTP Content Negotiation (`Accept` header). Clients and servers MUST NOT rely on the order in which properties appear in JSON. To keep the response size small, lists of resources (e.g. the list of batch jobs) usually should not include nested JSON objects, if this information can be requested from the individual resource endpoints (e.g. the metadata for a single batch job).

Charset

If not negotiated otherwise with HTTP Content Negotiation (`Accept-Charset` header), services use UTF-8 as the default charset.

Web Linking

The API is designed in a way that for most resources, such as collections and processes, a set of links can be added. These can be alternate representations such as data discovery via OGC WCS or OGC CSW, references to a license, references to actual raw data for downloading, detailed information about pre-processing and more. Clients should allow users to follow the links.

Whenever links are utilized in the API, the description explains which relation (`rel` property) types are commonly used. A list of standardized link relations types is provided by IANA and the API tries to align whenever feasible.

Some very common relation types - usually not mentioned explicitly in the description of `links` fields - are:

1. `self`: Refers to the location that the resource can be (permanently) found online. This is particularly useful when the data is made available offline, so that the downstream user knows the source of the data.

2. `alternate`: An alternative representation of the resource, may it be another metadata standard the data is available in or simply a human-readable version in HTML or PDF.

3. `about`: A resource that is related to or further explains the resource, e.g. a user guide.

4. `canonical`: This relation type usually points to a publicly accessible and more long-lived URL for a resource that otherwise often requires (Bearer) authentication with a short-lived token. This way the exposed resources can be used by non-openEO clients without additional authentication steps. For example, a shared user-defined process or batch job results could be exposed via a canonical link. If a URL should be publicly available to everyone, it can simply a user-specific URL, e.g. `https://openeo.example/processes/john_doe/ndvi`. For resources that should only be accessible to a certain group of user, a signed URL could be given, e.g. `https://openeo.example/processes/81zjh1tc2pt52gbx/ndvi`.

Generally, it is RECOMMENDED adding descriptive titles (property `title`) and media type information (property `type`) for a better user experience.

Error Handling

The success of requests MUST be indicated using HTTP status codes according to RFC 7231.

If the API responds with a status code between 100 and 399 the back-end indicates that the request has been handled successfully.

In general an error is communicated with a status code between 400 and 599. Client errors are defined as a client passing invalid data to the service and the service *correctly* rejecting that data. Examples include invalid credentials, incorrect parameters, unknown versions, or similar. These are generally "4xx" HTTP error codes and are the result of a client passing incorrect or invalid data. Client errors do *not* contribute to overall API availability.

Server errors are defined as the server failing to correctly return in response to a valid client request. These are generally "5xx" HTTP error codes. Server errors *do* contribute to the overall API availability. Calls that fail due to rate limiting or quota failures MUST NOT count as server errors.

JSON error object

A JSON error object SHOULD be sent with all responses that have a status code between 400 and 599.

```
{
  "id": "936DA01F-9ABD-4D9D-80C7-02AF85C822A8",
  "code": "SampleError",
  "message": "A sample error message.",
  "url": "https://openeo.example/docs/errors/SampleError"
}
```

Sending `code` and `message` is REQUIRED.

- A back-end MAY add a free-form `id` (unique identifier) to the error response to be able to log and track errors with further non-disclosable details.
- The `code` is either one of the standardized textual openEO error codes or a proprietary error code.
- The `message` explains the reason the server is rejecting the request. For "4xx" error codes the message explains how the client needs to modify the request.

  By default, the message MUST be in English. Content Negotiation is used to localize the error messages: If an `Accept-Language` header is sent by the client and a translation is available, the message should be translated accordingly, and the `Content-Language` header must be present in the response. See "How to localize your API" for more information.
- `url` is an OPTIONAL attribute and contains a link to a resource that is explaining the error and potential solutions in-depth.

Standardized status codes

The openEO API usually uses the following HTTP status codes for successful requests:

- **200 OK**: Indicates a successful request **with** a response body being sent.
- **201 Created** Indicates a successful request that successfully created a new resource. Sends a `Location` header to the newly created resource **without** a response body.
- **202 Accepted** Indicates a successful request that successfully queued the creation of a new resource, but it has not been created yet. The response is sent **without** a response body.
- **204 No Content**: Indicates a successful request **without** a response body being sent.

The openEO API reuses commonly used HTTP status codes for failed requests:

- **400 Bad Request**: The back-end responds with this error code whenever the error has its origin on client side and no other HTTP status code in the 400 range is suitable.
- **401 Unauthorized**: The client did not provide any authentication details for a resource requiring authentication or the provided authentication details are not correct.
- **403 Forbidden**: The client did provide correct authentication details, but the privileges/permissions of the provided credentials do not allow requesting the resource.
- **404 Not Found**: The resource specified by the path does not exist. One of the resources belonging to the specified identifiers are not available at the back-end. *Note:* Unsupported endpoints MAY also return an HTTP status code 501.
- **500 Internal Server Error**: The error has its origin on server side and no other status code in the 500 range is suitable.
- **501 Not Implemented**: The requested endpoint is specified by the openEO API, but is not implemented (yet) by the back-end. *Note:* Unsupported endpoints MAY also return HTTP status code 404.

If a HTTP status code in the 400 range is returned, the client SHOULD modify the request and repeat the request. For HTTP status code in the 500 range, the client MAY repeat the same request later.

All HTTP status codes defined in RFC 7231 in the 400 and 500 ranges can be used as openEO error code in addition to the most used status codes mentioned here. Responding with openEO error codes 400 and 500 SHOULD be avoided in favor of any more specific standardized or proprietary openEO error code.

Temporal data

Date, time, time intervals, and durations are formatted based on ISO 8601 or its profile RFC 3339 whenever there is an appropriate encoding available in the standard. All temporal data are specified based on the Gregorian calendar.

Authentication

The openEO API offers two forms of authentication by default:

- OpenID Connect (recommended) at `GET /credentials/oidc`
- Basic at `GET /credentials/basic`

After authentication with any of the methods listed above, the tokens obtained during the authentication workflows can be sent to protected endpoints in subsequent requests.

Further authentication methods MAY be added by back-ends.

Bearer

Security Scheme Type: HTTP

HTTP Authorization Scheme: `bearer`

Bearer format: `The Bearer Token MUST consist of the authentication method, a provider ID (if available) and the token itself. All separated by a forward slash `/`. Examples (replace `TOKEN` with the actual access token): (1) Basic authentication (no provider ID available): `basic//TOKEN` (2) OpenID Connect (provider ID is `ms`): `oidc/ms/TOKEN`. For OpenID Connect, the provider ID corresponds to the value specified for `id` for each provider in `GET /credentials/oidc`.`

Basic

Security Scheme Type: HTTP

HTTP Authorization Scheme: `basic`

Note: Although it is possible to request several public endpoints for capabilities and discovery that do not require authorization, it is RECOMMENDED that clients (re-)request the public endpoints that support Bearer authentication with the Bearer token once available to also retrieve any private data that is made available specifically for the authenticated user. This may require that clients clear any cached data they retrieved from public endpoints before.

Cross-Origin Resource Sharing (CORS)

> Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources [...] on a web page to be requested from another domain outside the domain from which the first resource was served. [...] CORS defines a way in which a browser and server can interact to determine whether it is safe to allow the cross-origin request. This allows for more freedom and functionality than purely same-origin requests, but is more secure than simply allowing all cross-origin requests.

Source: https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

openEO-based back-ends are usually hosted on a different domain / host than the client that is requesting data from the back-end. Therefore, most requests to the back-end are blocked by all modern browsers. This leads to the problem that the JavaScript library and any browser-based application can not access back-ends. Therefore, all back-end providers SHOULD support CORS to enable browser-based applications to access back-ends. CORS is a recommendation of the W3C organization. The following chapters explain how back-end providers can implement CORS support.

Tip: Most servers can send the required headers and the responses to the OPTIONS requests automatically for all endpoints. Otherwise, a proxy server may be used to add the headers and OPTIONS responses.

CORS headers

The following headers MUST be included with every response:

Name	Description	Example
Access-Control-Allow-Origin	Allowed origin for the request, including protocol, host and port or `*` for all origins. Returning the value `*` to allow requests from browser-based implementations is RECOMMENDED.	`*`
Access-Control-Expose-Headers	Some endpoints require sending additional HTTP response headers such as `OpenEO-Identifier` and `Location`. To make these headers available to browser-based clients, they MUST be white-listed with this CORS header. The following HTTP headers are white-listed by browsers and MUST NOT be included: `Cache-Control`, `Content-Language`, `Content-Length`, `Content-Type`, `Expires`, `Last-Modified` and `Pragma`. At least the following headers MUST be listed in this version of the openEO API: `Link`, `Location`, `OpenEO-Costs` and `OpenEO-Identifier`.	`Link, Location, OpenEO-Costs, OpenEO-Identifier`

Example request and response

Request:

```
POST /api/v1/jobs HTTP/1.1
Host: openeo.example
Origin: https://company.example:8080
Authorization: Bearer basic//ZXhhbXBsZTpleGFtcGxl
```

Response:

```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Location, OpenEO-Identifier, OpenEO-Costs, Link
Content-Type: application/json
Location: https://openeo.example/api/v1/jobs/abc123
OpenEO-Identifier: abc123
```

OPTIONS method

All endpoints defined in the API specification must additionally respond to the `OPTIONS` HTTP method. This is a response for the preflight requests made by web browsers before sending the actual request (e.g. `POST /jobs`). It needs to respond with a status code of `204` and no response body. **In addition** to the HTTP headers shown in the table above, the following HTTP headers MUST be included with every response to an `OPTIONS` request:

Name	Description	Example
Access-Control-Allow-Headers	Comma-separated list of HTTP headers allowed to be sent with the actual (non-preflight) request. MUST contain at least `Authorization` if any kind of authorization is implemented by the back-end.	`Authorization, Content-Type`
Access-Control-Allow-Methods	Comma-separated list of HTTP methods allowed to be requested. Back-ends MUST list all implemented HTTP methods for the endpoint.	`OPTIONS, GET, POST, PATCH, PUT, DELETE`
Content-Type	SHOULD return the content type delivered by the request that the permission is requested for.	`application/json`

Example request and response

Request:

```
OPTIONS /api/v1/jobs HTTP/1.1
Host: openeo.example
Origin: https://company.example:8080
Access-Control-Request-Method: POST
Access-Control-Request-Headers: Authorization, Content-Type
```

Note that the `Access-Control-Request-*` headers are automatically attached to the requests by the browsers.

Response:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: OPTIONS, GET, POST, PATCH, PUT, DELETE
Access-Control-Allow-Headers: Authorization, Content-Type
Access-Control-Expose-Headers: Location, OpenEO-Identifier, OpenEO-Costs, Link
Content-Type: application/json
```

Processes

A **process** is an operation that performs a specific task on a set of parameters and returns a result. An example is computing a statistical operation, such as mean or median, on selected EO data. A process is similar to a function or method in programming languages. In openEO, processes are used to build a chain of processes (process graph), which can be applied to EO data to derive your own findings from the data.

A **predefined process** is a process provided by the *back-end*. There is a set of predefined processes by openEO to improve interoperability between back-ends. Back-ends SHOULD follow these specifications whenever possible. Not all processes need to be implemented by all back-ends. See the **process reference** for predefined processes.

A **user-defined process** is a process defined by the *user*. It can directly be part of another process graph or be stored as custom process on a back-end. Internally, it is a *process graph* with a variety of additional metadata.

A **process graph** chains specific process calls from the set of predefined and user-defined processes together. A process graph itself can be stored as a (user-defined) process again. Similarly to scripts in the context of programming, process graphs organize and automate the execution of one or more processes that could alternatively be executed individually. In a process graph, processes need to be specific, i.e. concrete values or "placeholders" for input parameters need to be specified. These values can be scalars, arrays, objects, references to parameters or previous computations or other process graphs.

Defining Processes

Back-ends and users MAY define new proprietary processes for their domain.

Back-end providers MUST follow the schema for predefined processes as in `GET /processes` to define new processes. This includes:

- Choosing a intuitive process id, consisting of only letters (a-z), numbers and underscores. It MUST be unique across the predefined processes.
- Defining the parameters and their exact (JSON) schemes.
- Specifying the return value of a process also with a (JSON) schema.
- Providing examples or compliance tests.
- Trying to make the process universally usable so that other back-end providers or openEO can adopt it.

Users MUST follow the schema for user-defined processes as in `GET /process_graphs/{process_graph_id}` to define new processes. This includes:

- Choosing a intuitive name as process id, consisting of only letters (a-z), numbers and underscores. It MUST be unique per user across the user-defined processes.
- Defining the algorithm as a process graph.
- Optionally, specifying the additional metadata for processes.

If new process are potentially useful for other back-ends the openEO consortium is happily accepting pull requests to include them in the list of predefined processes.

Schemas

Each process parameter and the return values of a process define a schema that the value MUST comply to. The schemas are based on JSON Schema draft-07.

Multiple custom keywords have been defined:

- `subtype` for more fine-grained data-types than JSON Schema supports.
- `dimensions` to further define the dimension types required if the `subtype` is `datacube`.
- `parameters` to specify the parameters of a process graph if the `subtype` is `process-graph`.
- `returns` to describe the return value of a process graph if the `subtype` is `process-graph`.

Subtypes

JSON Schema allows to specify only a small set of native data types (string, boolean, number, integer, array, object, null). To support more fine grained data types, a custom JSON Schema keyword has been defined: `subtype`. It works similarly as the JSON Schema keyword `format` and standardizes a number of openEO related data types that extend the native data types, for example: `bounding-box` (object with at least `west`, `south`, `east` and `north` properties), `date-time` (string representation of date and time following RFC 3339), `datacube` (a datacube with dimensions), etc. The subtypes should be re-used in process schema definitions whenever suitable.

If a general data type such as `string` or `number` is used in a schema, all subtypes with the same parent data type can be passed, too. Clients should offer make passing subtypes as easy as passing a general data type. For example, a parameter accepting strings must also allow passing a string with subtype `date` and thus clients should encourage this by also providing a date-picker.

A list of predefined subtypes is available as JSON Schema in openeo-processes.

Process Graphs

As defined above, a **process graph** is a chain of processes with explicit values for their parameters. Technically, a process graph is defined to be a graph of connected processes with exactly one node returning the final result:

```
<ProcessGraph> := {
  "<ProcessNodeIdentifier>": <ProcessNode>,
  ...
}
```

`<ProcessNodeIdentifier>` is a unique key within the process graph that is used to reference (the return value of) this process in arguments of other processes. The identifier is unique only strictly within itself, excluding any parent and child process graphs. Process node identifiers are also strictly scoped and can not be referenced from child or parent process graphs. Circular references are not allowed.

Note: We provide a non-binding JSON Schema for basic process graph validation.

Processes (Process Nodes)

A single node in a process graph (i.e. a specific instance of a process) is defined as follows:

```
<ProcessNode> := {
  "process_id": <string>,
  "namespace": <string> / null,
  "description": <string>,
  "arguments": <Arguments>,
  "result": true / false
}
```

A process node MUST always contain key-value-pairs named `process_id` and `arguments`. It MAY contain a `description`.

One of the nodes in a map of processes (the final one) MUST have the `result` flag set to `true`, all the other nodes can omit it as the default value is `false`. Having such a node is important as multiple end nodes are possible, but in most use cases it is important to exactly specify the return value to be used by other processes. Each child process graph must also specify a result node similar to the "main" process graph.

`process_id` MUST be a valid process ID in the `namespace` given. Clients SHOULD warn the user if a user-defined process is added with the same identifier as one of the predefined process.

Arguments

A process can have an arbitrary number of arguments. Their name and value are specified in the process specification as an object of key-value pairs:

```
<Arguments> := {
  "<ParameterName>": <string|number|boolean|null|array|object|ResultReference|
}
```

Notes:

- The specified data types are the native data types supported by JSON, except for `ResultReference`, `UserDefinedProcess` and `ParameterReference`.
- Objects are not allowed to have keys with the following reserved names:
  - `from_node`, except for objects of type `ResultReference`
  - `process_graph`, except for objects of type `UserDefinedProcess`
  - `from_parameter`, except for objects of type `ParameterReference`
- Arrays and objects can also contain a `ResultReference`, a `UserDefinedProcess` or a `ParameterReference`. So back-ends must *fully* traverse the process graphs, including all children.

Accessing results of other process nodes

A value of type `<ResultReference>` is an object with a key `from_node` and a `<ProcessNodeIdentifier>` as corresponding value:

```
<ResultReference> := {
  "from_node": "<ProcessNodeIdentifier>"
}
```

This tells the back-end that the process expects the result (i.e. the return value) from another process node to be passed as argument. The `<ProcessNodeIdentifier>` is strictly scoped and can only reference nodes from within the same process graph, not child or parent process graphs.

Child processes

Some processes can run child processes, which is similar to the concept that other programming languages call callbacks or lambda functions. Each child process is simply a user-defined process again and can in theory be arbitrarily complex.

A very simple example would be to calculate the absolute value of each pixel in a data cube. This can be achieved in openEO by using the `apply` process which gets the `absolute` process passed as child process. In this example, the "child" processes consists of a single process `absolute`, but it can also be a more complex computation such as an NDVI or a prediciton based on a machine learning model.

Example:

A `<UserDefinedProcess>` argument MUST at least consist of an object with a key `process_graph`. Optionally, it can also be described with the same additional properties available for predefined processes such as an id, parameters, return values etc. When embedded in a process graph, these additional properties of a user-defined process are usually not used, except for validation purposes.

```
<UserDefinedProcess> := {
  "process_graph": <ProcessGraph>,
  ...
}
```

Accessing process parameters

A "parent" process that works with a child process can make so called *process graph parameters* available to the "child" logic. Processes in the "child" process graph can access these parameters by passing a `ParameterReference` object as argument. It is an object with key `from_parameter` specifying the name of the process graph parameter:

```
<ParameterReference> := {
  "from_parameter": "<ParameterReferenceName>"
}
```

The parameter names made available for `<ParameterReferenceName>` are defined and passed to the process graph by one of the parent entities. The parent could be a process (such as `apply` or `reduce_dimension`) or something else that executes a process graph (a secondary web service for example). If the parent is a process, the parameter are defined in the `parameters` property of the corresponding JSON Schema.

In case of the example given above, the parameter `process` in the process `apply` defines two process graph parameters: `x` (the value of each pixel that will be processed) and `context` (additional data passed through from the user). The process `absolute` expects an argument with the same name `x`. The process graph for the example would look as follows:

```
{
  "process_id": "apply",
  "arguments": {
    "data": {"from_node": "loadcollection1"}
    "process": {
      "process_graph": {
        "abs1": {
          "process_id": "absolute",
          "arguments": {
            "x": {"from_parameter": "x"}
          },
          "result": true
        }
      }
    }
  }
}
```

`loadcollection1` would be a result from another process, which is not part of this example.

Important: `<ParameterReferenceName>` is less strictly scoped than `<ProcessNodeIdentifier>`. `<ParameterReferenceName>` can be any parameter from the process graph or any of its parents.

The value for the parameter MUST be resolved as follows:

1. In general the most specific parameter value is used. This means the parameter value is resolved starting from the current scope and then checking each parent for a suitable parameter value until a parameter values is found or the "root" process graph has been reached.
2. In case a parameter value is not available, the most unspecific default value from the process graph parameter definitions are used. For example, if default values are available for the root process graph and all children, the default value from the root process graph is used.
3. If no default values are available either, the error `ProcessParameterMissing` must be thrown.

Full example for an EVI computation

Deriving minimum EVI (Enhanced Vegetation Index) measurements over pixel time series of Sentinel-2 imagery. The main process graph in blue, child process graphs in yellow:

Graph with processing instructions

The process graph for the algorithm: pg-evi-example.json

Data Processing

Processes can run in three different ways:

1. Results can be pre-computed by creating a ***batch job***. They are submitted to the back-end's processing system, but will remain inactive until explicitly put into the processing queue. They will run only once and store results after execution. Results can be downloaded. Batch jobs are typically time consuming and user interaction is not possible although log files are generated for them. This is the only mode that allows to get an estimate about time, volume and costs beforehand.

2. A more dynamic way of processing and accessing data is to create a **secondary web service**. They allow web-based access using different protocols such as OGC WMS, OGC WCS, OGC API - Features or XYZ tiles. Some protocols such as the OGC WMS or XYZ tiles allow users to change the viewing extent or level of detail (zoom level). Therefore, computations often run *on demand* so that the requested data is calculated during the request. Back-ends should make sure to cache processed data to avoid additional/high costs and reduce waiting times for the user.

3. Processes can also be executed **on-demand** (i.e. synchronously). Results are delivered with the request itself and no job is created. Only lightweight computations, for example previews, should be executed using this approach as timeouts are to be expected for long-polling HTTP requests.

Validation

Process graph validation is a quite complex task. There's a JSON schema for basic process graph validation. It checks the general structure of a process graph, but only checking against the schema is not fully validating a process graph. Note that this JSON Schema is probably good enough for a first version, but should be revised and improved for production. There are further steps to do:

1. Validate whether there's exactly one `result: true` per process graph.
2. Check whether the process names that are referenced in the field `process_id` are actually available in the corresponding `namespace`.
3. Validate all arguments for each process against the JSON schemas that are specified in the corresponding process specifications.
4. Check whether the values specified for `from_node` have a corresponding node in the same process graph.
5. Validate whether the return value and the arguments requesting a return value with `from_node` are compatible.
6. Check the content of arrays and objects. These could include parameter and result references (`from_node`, `from_parameter` etc.).

Execution

To process the process graph on the back-end you need to go through all nodes/processes in the list and set for each node to which node it passes data and from which it expects data. In another iteration the back-end can find all start nodes for processing by checking for zero dependencies.

You can now start and execute the start nodes (in parallel, if possible). Results can be passed to the nodes that were identified beforehand. For each node that depends on multiple inputs you need to check whether all dependencies have already finished and only execute once the last dependency is ready.

Please be aware that the result node (`result` set to `true`) is not necessarily the last node that is executed. The author of the process graph may choose to set a non-end node to the result node!

Capabilities

General information about the API implementation and other supported capabilities provided by the back-end.

Information about the back-end

GET /

Lists general information about the back-end, including which version and endpoints of the openEO API are supported. May also include billing information.

AUTHORIZATIONS: None

Responses

200 Information about the API version and supported endpoints / features.

This endpoint MUST always be available for the API to be valid.

RESPONSE SCHEMA: application/json

`api_version` required	string Value: `"1.2.0"` Version number of the openEO specification the back-end implements.
`backend_version` required	string Version number of the back-end implementation. Every change on back-end side MUST cause a change of the version number.
`stac_version` required	string (stac_version) `^(0\.9.\d+|1\.\d+.\d+)` The version of the STAC specification, which MAY not be equal to the STAC API version. Supports versions 0.9.x and 1.x.x.
`type`	string Value: `"Catalog"` For STAC versions >= 1.0.0-rc.1 this field is required.
`id` required	string Identifier for the service. This field originates from STAC and is used as unique identifier for the STAC catalog available at `/collections`.
`title` required	string The name of the service.
`description` required	string <commonmark> A description of the service, which allows the service provider to introduce the user to its service. CommonMark 0.29 syntax MAY be used for rich text representation.
`conformsTo`	Array of strings <uri> (conformsTo) [items <uri >] Lists all conformance classes specified in various standards that the implementation conforms to. Conformance classes are commonly used in all OGC API standards and the STAC API specification. The general openEO conformance class is `https://api.openeo.org/1.2.0`. See the individual openEO API extensions for their conformance classes. This property is REQUIRED for STAC API versions 1.0.0-beta.1 and later.
`production`	boolean (production) Default: `false` Specifies whether the implementation is ready to be used in production use (`true`) or not (`false`). Clients SHOULD only connect to non-production implementations if the user explicitly confirmed to use a non-production implementation. This flag is part of `GET /.well-known/openeo` and `GET /`. It MUST be used consistently in both endpoints.
`endpoints` › required	Array of objects (Endpoint) Lists all supported endpoints. Supported are all endpoints, which are implemented, return usually a 2XX or 3XX HTTP status code and are fully compatible to the API specification. An entry for this endpoint (path `/` with method `GET`) SHOULD NOT be listed. Each path MUST only be listed once in the array.
`billing` ›	object (Billing) Billing related data, e.g. the currency used or available plans to process jobs. This property MUST be specified if the back-end uses any billing related API functionalities, e.g. budgeting or estimates. The absence of this property does not mean the back-end is necessarily free to use for all. Providers may choose to bill users outside of the API, e.g. with a monthly fee that is not depending on individual API interactions.
`links` › required	Array of objects (Link) Links related to this service, e.g. the homepage of the service provider or the terms of service. It is highly RECOMMENDED to provide links with the following `rel` (relation) types: 1. `version-history`: A link back to the Well-Known URL (including `/.well-known/openeo`, see the corresponding endpoint for details) to allow clients to work on the most recent version. 2. `terms-of-service`: A link to the terms of service. If a back-end provides a link to the terms of service, the clients MUST provide a way to read the terms of service and only connect to the back-end after the user agreed to them. The user interface MUST be designed in a way that the terms of service are not agreed to by default, i.e. the user MUST explicitly agree to them. 3. `privacy-policy`: A link to the privacy policy (GDPR). If a back-end provides a link to a privacy policy, the clients MUST provide a way to read the privacy policy and only connect to the back-end after the user agreed to them. The user interface MUST be designed in a way that the privacy policy is not agreed to by default, i.e. the user MUST explicitly agree to them. 4. `service-desc` or `service-doc`: A link to the API definition. Use `service-desc` for machine-readable API definition and `service-doc` for human-readable API definition. Required if full OGC API compatibility is desired. 5. `conformance`: A link to the Conformance declaration (see `/conformance`). Required if full OGC API compatibility is desired. 6. `data`: A link to the collections (see `/collections`). Required if full OGC API compatibility is desired. 7. `create-form`: A link to a user registration page. 8. `recovery-form`: A link to a page where a user can recover a user account (e.g. to reset the password or send a reminder about the username to the user's email account). For additional relation types see also the lists of common relation types in openEO.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request usually does not respond with HTTP status codes 401 and 403 due to missing authorization. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "api_version": "1.2.0",
    "backend_version": "1.1.2",
    "stac_version": "1.0.0",
    "type": "Catalog",
    "id": "cool-eo-cloud",
    "title": "Example Cloud Corp.",
    "description": "This service is provided t
  - "conformsTo": [
        "https://api.openeo.org/1.2.0",
        "https://api.openeo.org/extensions/comm
        "https://api.openeo.org/extensions/fede
        "https://api.stacspec.org/v1.0.0/core",
        "https://api.stacspec.org/v1.0.0/colle
    ],
    "production": false,
  - "endpoints": [
      + { … },
      + { … },
      + { … },
      + { … },
      + { … }
    ],
  - "billing": {
        "currency": "USD",
        "default_plan": "free",
      + "plans": [ … ]
    },
  - "links": [
      + { … },
      + { … },
      + { … },
      + { … },
      + { … },
      + { … },
      + { … },
      + { … }
    ]
}
```

Supported openEO versions

GET /.well-known/openeo

Lists all implemented openEO versions supported by the service provider. This endpoint is the Well-Known URI (see RFC 5785) for openEO.

This allows a client to easily identify the most recent openEO implementation it supports. By default, a client SHOULD connect to the most recent production-ready version it supports. If not available, the most recent supported version of *all* versions SHOULD be connected to. Clients MAY let users choose to connect to versions that are not production-ready or outdated. The most recent version is determined by comparing the version numbers according to rules from Semantic Versioning, especially §11. Any pair of API versions in this list MUST NOT be equal according to Semantic Versioning.

The Well-Known URI is the entry point for clients and users, so make sure it is permanent and easy to use and remember. Clients MUST NOT require the well-known path (`/.well-known/openeo`) in the URL that is specified by a user to connect to the back-end.

For clients, the usual behavior SHOULD follow these steps:

1. The user provides a URI, which may consist of a scheme (protocol), an authority (host, port) and a path.
2. The client parses the URI and appends `/.well-knwon/openeo` to the path. Make sure to correctly handle leading/trailing slashes.
3. Send a request to the new URI. A. On success: Detect the most suitable API instance/version (see above) and read the capabilities from there. B. On failure: Directly try to read the capabilities from the original URI given by the user.

This URI MUST NOT be versioned as the other endpoints. If your API is available at `https://openeo.example/api/v1`, and you instruct your API users to use `https://openeo.example` as connection URI, the Well-Known URI SHOULD be located at `https://openeo.example/.well-known/openeo`. The Well-Known URI is usually directly located at the top-level, but it is not a requirement. For example, `https://openeo.example/eo/.well-known/openeo` is also allowed.

Clients MAY get additional information (e.g. title or description) about a back-end from the most recent version that has the `production` flag set to `true`.

AUTHORIZATIONS: None

Responses

200 List of all available API instances, each with URL and the implemented openEO API version.

RESPONSE SCHEMA: application/json

`versions` › required	Array of objects (API Instance)

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request usually does not respond with HTTP status codes 401 and 403 due to missing authorization. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "versions": [
      + { … },
      + { … },
      + { … }
    ]
}
```

Supported file formats

GET /file_formats

Lists supported input and output file formats. *Input* file formats specify which file a back-end can *read* from. *Output* file formats specify which file a back-end can *write* to.

The response to this request is an object listing all available input and output file formats separately with their parameters and additional data. This endpoint does not include the supported secondary web services.

Note: Format names and parameters MUST be fully aligned with the GDAL codes if available, see GDAL Raster Formats and OGR Vector Formats. It is OPTIONAL to support all output format parameters supported by GDAL. Some file formats not available through GDAL may be defined centrally for openEO. Custom file formats or parameters MAY be defined.

The format descriptions MUST describe how the file formats relate to data cubes. Input file formats MUST describe how the files have to be structured to be transformed into data cubes. Output file formats MUST describe how the data cubes are stored at the back-end and how the resulting file structure looks like.

Back-ends MUST NOT support aliases, for example it is not allowed to support `geotiff` instead of `gtiff`. Nevertheless, openEO Clients MAY translate user input for convenience (e.g. translate `geotiff` to `gtiff`). Also, for a better user experience the back-end can specify a `title`.

Format names MUST be accepted in a *case insensitive* manner throughout the API.

AUTHORIZATIONS: None or *Bearer*

Responses

200
An object with containing all input and output format separately. For each property `input` and `output` an object is defined where the file format names are the property keys and the property values are objects that define a title, supported parameters and related links.

RESPONSE SCHEMA: application/json

`input` › required	object (Input File Formats) Map of supported input file formats, i.e. file formats a back-end can **read** from. The property keys are the file format names that are used by clients and users, for example in process graphs.
`output` › required	object (Output File Formats) Map of supported output file formats, i.e. file formats a back-end can **write** to. The property keys are the file format names that are used by clients and users, for example in process graphs.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request usually does not respond with HTTP status codes 401 and 403 due to missing authorization. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "output": {
      + "GTiff": { … },
      + "GPKG": { … }
    },
  - "input": {
      + "GPKG": { … }
    }
}
```

Conformance classes this API implements

GET /conformance

Lists all conformance classes specified in various standards that the implementation conforms to. Conformance classes are commonly used in all OGC API standards and the STAC API specification. openEO adds relatively broadly defined conformance classes, especially for the extensions. Otherwise, the implemented functionality can usually be retrieved from the capabilities in openEO.

The general openEO conformance class is `https://api.openeo.org/1.2.0`. See the individual openEO API extensions for their conformance classes.

The conformance classes listed at this endpoint and listed in the corresponding `conformsTo` property in `GET /` MUST be equal.

More details:

- STAC API, especially the conformance class "STAC API - Collections"
- OGC API standards

Responses

200 The URIs of all conformance classes supported by the server.

RESPONSE SCHEMA: application/json

`conformsTo` required	Array of strings <uri> (conformsTo) [items <uri >] Lists all conformance classes specified in various standards that the implementation conforms to. Conformance classes are commonly used in all OGC API standards and the STAC API specification. The general openEO conformance class is `https://api.openeo.org/1.2.0`. See the individual openEO API extensions for their conformance classes. This property is REQUIRED for STAC API versions 1.0.0-beta.1 and later.

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "conformsTo": [
        "https://api.openeo.org/1.2.0",
        "https://api.openeo.org/extensions/comm
        "https://api.openeo.org/extensions/fede
        "https://api.stacspec.org/v1.0.0/core",
        "https://api.stacspec.org/v1.0.0/colle
    ]
}
```

Supported UDF runtimes

GET /udf_runtimes

Lists the supported runtimes for user-defined functions (UDFs), which includes either the programming languages including version numbers and available libraries including version numbers or docker containers.

AUTHORIZATIONS: None or *Bearer*

Responses

200 Description of UDF runtime support

RESPONSE SCHEMA: application/json

`UDF Runtime name*` › additional property	object

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "PHP": {
        "title": "PHP v7.x",
        "description": "Just an example how to
        "experimental": true,
        "type": "docker",
        "docker": "openeo/udf-php7",
        "default": "latest",
      + "tags": [ … ],
      + "links": [ … ]
    },
  - "R": {
        "title": "R v3.x for Statistical Comput
        "description": "R programming language
        "type": "language",
        "default": "3.5.2",
      + "versions": { … }
    }
}
```

Supported secondary web service protocols

GET /service_types

Lists supported secondary web service protocols such as OGC WMS, OGC WCS, OGC API - Features or XYZ tiles. The response is an object of all available secondary web service protocols with their supported configuration settings and expected process parameters.

- The configuration settings for the service SHOULD be defined upon creation of a service and the service will be set up accordingly.
- The process parameters SHOULD be referenced (with a `from_parameter` reference) in the user-defined process that is used to compute web service results. The appropriate arguments MUST be provided to the user-defined process, usually at runtime from the context of the web service. For example, a map service such as a WMS would need to inject the spatial extent into the user-defined process so that the back-end can compute the corresponding tile correctly.

To improve interoperability between back-ends common names for the services SHOULD be used, e.g. the abbreviations used in the official OGC Schema Repository for the respective services.

Service names MUST be accepted in a *case insensitive* manner throughout the API.

AUTHORIZATIONS: None or *Bearer*

Responses

200
An object with a map containing all service names as keys and an object that defines supported configuration settings and process parameters.

RESPONSE SCHEMA: application/json

`Service Name*` › additional property	object (Service Type)

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request usually does not respond with HTTP status codes 401 and 403 due to missing authorization. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "WMS": {
        "title": "OGC Web Map Service",
      + "configuration": { … },
      + "process_parameters": [ … ],
      + "links": [ … ]
    },
  - "OGCAPI-FEATURES": {
        "title": "OGC API - Features",
        "description": "Exposes a OGC API - Fe
      + "configuration": { … },
        "process_parameters": [ ],
      + "links": [ … ]
    }
}
```

Account Management

The following endpoints handle user profiles, accounting and authentication. See also Authentication. In general, the openEO API only defines a minimum subset of user management and accounting functionality. It allows to

- authenticate and authorize a user, which may include user registration with OpenID Connect,
- handle storage space limits (disk quota),
- manage billing, which includes to
  - query the credit a user has available,
  - estimate costs for certain operations (data processing and downloading),
  - get information about produced costs,
  - limit costs of certain operations.

Therefore, the API leaves some aspects open that have to be handled by the back-ends separately, including

- credentials recovery, e.g. retrieving a forgotten password
- user data management, e.g. changing the users payment details or email address
- payments, i.e. topping up credits for pre-paid services or paying for post-paid services
- other accounting related tasks, e.g. creating invoices,
- user registration (except for user registration with OpenID Connect).

OpenID Connect authentication

GET /credentials/oidc

Lists the supported OpenID Connect providers (OP). OpenID Connect Providers MUST support OpenID Connect Discovery.

It is highly RECOMMENDED to implement OpenID Connect for public services in favor of Basic authentication.

openEO clients MUST use the **access token** as part of the Bearer token for authorization in subsequent API calls (see also the information about Bearer tokens in this document). Clients MUST NOT use the id token or the authorization code. The access token provided by an OpenID Connect Provider does not necessarily provide information about the issuer (i.e. the OpenID Connect provider) and therefore a prefix MUST be added to the Bearer Token sent in subsequent API calls to protected endpoints. The Bearer Token sent to protected endpoints MUST consist of the authentication method (here `oidc`), the provider ID and the access token itself. All separated by a forward slash `/`. The provider ID corresponds to the value specified for `id` for each provider in the response body of this endpoint. The header in subsequent API calls for a provider with `id` `ms` would look as follows: `Authorization: Bearer oidc/ms/TOKEN` (replace `TOKEN` with the actual access token received from the OpenID Connect Provider).

Back-ends MAY request user information (including Claims) from the OpenID Connect Userinfo endpoint using the access token (without the prefix described above). Therefore, both openEO client and openEO back-end are relying parties (clients) to the OpenID Connect Provider.

AUTHORIZATIONS: None

Responses

200 Lists the OpenID Connect Providers.

RESPONSE SCHEMA: application/json

`providers` › required	Array of objects (OpenID Connect Provider) non-empty The first provider in this list is the default provider for authentication. Clients can either pre-select or directly use the default provider for authentication if the user does not specify a specific value.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "providers": [
      + { … },
      + { … },
      + { … }
    ]
}
```

HTTP Basic authentication

GET /credentials/basic

Checks the credentials provided through HTTP Basic Authentication according to RFC 7617 and

Response samples

200 4XX 5XX

Content type
application/json

returns an access token for valid credentials.

The credentials (username and password) MUST be sent in the HTTP header `Authorization` with type `Basic` and the Base64 encoded string consisting of username and password separated by a double colon `:`. The header would look as follows for username `user` and password `pw`: `Authorization: Basic dXNlcjpwdw==`.

The access token has to be used in the Bearer token for authorization in subsequent API calls (see also the information about Bearer tokens in this document). The access token returned by this request MUST NOT be provided with `basic//` prefix, but the Bearer Token sent in subsequent API calls to protected endpoints MUST be prefixed with `basic//`. The header in subsequent API calls would look as follows: `Authorization: Bearer basic//TOKEN` (replace `TOKEN` with the actual access token).

It is RECOMMENDED to implement this authentication method for non-public services only.

AUTHORIZATIONS: > *Basic*

Responses

200 Credentials are correct and authentication succeeded.

RESPONSE SCHEMA: application/json

`access_token` required	string The access token (without `basic//` prefix) to be used in the Bearer token for authorization in subsequent API calls.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Copy

```
{
    "access_token": "b34ba2bdf9ac9ee1"
}
```

Information about the authenticated user

GET `/me`

Lists information about the authenticated user such as the user id. The endpoint MAY return the disk quota available to the user. The endpoint MAY also return links related to user management and the user profile, e.g. where payments are handled or the user profile could be edited. For back-ends that involve accounting, this service MAY also return the currently available money or credits in the currency the back-end is working with. This endpoint MAY be extended to fulfil the specification of the OpenID Connect UserInfo Endpoint.

AUTHORIZATIONS: > *Bearer*

Responses

200 Information about the logged in user.

RESPONSE SCHEMA: application/json

`user_id` required	string `^[\w\-\.~]+$` A unique user identifier specific to the back-end, which could either be chosen by a user or is automatically generated by the back-end during the registration process at the back-end. It is meant to be used as an identifier in URIs (e.g. for sharing purposes), which is primarily used in machine-to-machine communication. Preferrably use the human-readable property `name` to display the user's name in user interfaces instead of the user identifier.
`name`	string The user name, a human-friendly displayable name. Could be the user's real name or a nickname.
`default_plan`	string Name of the single plan the user is currently subscribed to if any.
`storage` >	object or null (User Storage) Information about the storage space available to the user.
`budget`	number or null The remaining budget a user has available. The value MUST be specified in the currency of the back-end. The value SHOULD be set to `null` if no explicit limit applies.
`links` >	Array of objects (Link) Links related to the user profile, e.g. where payments are handled or the user profile could be edited. Providing links with the following `rel` (relation) types is RECOMMENDED: 1. `payment`: A page where users can recharge their user account with money or credits. 2. `edit-form`: Points to a page where the user can edit his user profile. 3. `alternate`: Any other representation of these (and potentially additional) user information, e.g. the (public) user profile page. It is RECOMMENDED to add descriptive titles for a better user experience. 4. `related`: Any other user-specific links to be shown in clients, e.g. to user-specific settings, invoices, etc. It is RECOMMENDED to add descriptive titles for a better user experience. For additional relation types see also the lists of common relation types in openEO.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "user_id": "john_doe",
    "name": "string",
    "default_plan": "free",
  - "storage": {
        "free": 536870912,
        "quota": 1073741824
    },
    "budget": 0,
  - "links": [
      + { … },
      + { … },
      + { … },
      + { … },
      + { … }
    ]
}
```

EO Data Discovery

These endpoints allow to list the collections that are available at the back-end and can be used as data cubes for data processing.

STAC

For data discovery of Earth Observation Collections at the back-ends, openEO strives for compatibility with the specifications SpatioTemporal Asset Catalog (STAC) and OGC API - Features - Part 1: Core as far as possible. Implementing the data discovery endpoints of openEO also produced valid STAC API 1.0 and OGC API - Features 1.0 responses, including (partial) compatibility with their APIs.

The data discovery endpoints `GET /collections` and `GET /collections/{collection_id}` are compatible with OGC API - Features and STAC. Both specifications define additional endpoints that need to be implemented to be fully compatible. The additional endpoints can easily be integrated into an openEO API implementation. A rough list of actions for compatibility is available below, but please refer to their specifications to find out the full details.

Important: STAC specification and STAC API are different specifications and have different version numbers after version 0.9.0. The openEO API implements STAC API versions >= 0.9.0 (preferrably STAC API version 1.0.0, but 0.9.0 is allowed for backward compatibility), which allows to serve all STAC specification versions in the range of 0.9.x and 1.x.x (see the `stac_version` property).

Content Extensions

STAC has several extensions that can be used to better describe your data. Clients and server are not required to implement all of them, so be aware that some clients may not be able to read all your metadata.

Some commonly used extensions that are relevant for datasets exposed through the openEO API are:

- Data Cube extension (part of the openEO API)
- EO (Electro-Optical) extension
- Processing extension
- Raster extension
- SAR extension
- Satellite extension
- Scientific Citation extension

Provide data for download

If you'd like to provide your data for download in addition to offering the cloud processing service, you can implement the full STAC API. Therefore, you can implement the endpoints `GET /collections/{collection_id}/items` and `GET /collections/{collection_id}/items/{feature_id}` to support retrieval of individual items. To benefit from the STAC ecosystem and allow searching for items you can also implement `POST /search` and `GET /search`. Further information can be found in the STAC API repository.

API Extensions

STAC API has several extensions that can be implemented on top of the openEO API to enrich the API functionality, e.g. for searching.

Basic metadata for all datasets

GET `/collections`

Lists available collections with at least the required information.

It is **strongly RECOMMENDED** to keep the response size small by omitting larger optional values from the objects in `collections` (e.g. the `summaries` and `cube:dimensions` properties). To get the full metadata for a collection clients MUST request `GET /collections/{collection_id}`.

This endpoint is compatible with STAC API 0.9.0 and later and OGC API - Features 1.0. STAC API extensions and STAC extensions can be implemented in addition to what is documented here.

Note: Although it is possible to request public collections without authorization, it is RECOMMENDED that clients (re-)request the collections with the Bearer token once available to also retrieve any private collections.

AUTHORIZATIONS: > None or *Bearer*

QUERY PARAMETERS

`limit`	integer >= 1 Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 Lists of collections and related links.

RESPONSE SCHEMA: application/json

`collections` > required	Array of objects (Collection)
`links` > required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "collections": [
      + { … },
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

Full metadata for a specific dataset

GET `/collections/{collection_id}`

Lists **all** information about a specific collection specified by the identifier `collection_id`.

This endpoint is compatible with STAC API 0.9.0 and later and OGC API - Features 1.0. STAC API extensions and STAC extensions can be implemented in addition to what is documented here.

Note: Providing the Bearer token is REQUIRED for private collections.

AUTHORIZATIONS: > None or *Bearer*

PATH PARAMETERS

`collection_id` required	string (collection_id) `^[\w\-\.~\/]+$` Example: `Sentinel-2A` Collection identifier

Responses

200 JSON object with the full collection metadata.

RESPONSE SCHEMA: application/json

`stac_version` required	string (stac_version) `^(0\.9.\d+|1\.\d+.\d+)` The version of the STAC specification, which MAY not be equal to the STAC API version. Supports versions 0.9.x and 1.x.x.
`stac_extensions` >	Array of Reference to a JSON Schema (string) or Reference to a core extension (STAC < 1.0.0-rc.1 only) (string) (stac_extensions) unique A list of implemented STAC extensions. The list contains URLs to the JSON Schema files it can be validated against. For STAC < 1.0.0-rc.1 shortcuts such as `sar` can be used instead of the schema URL.
`type`	string Value: `"Collection"` For STAC versions >= 1.0.0-rc.1 this field is required.
`id` required	string (collection_id) `^[\w\-\.~\/]+$` A unique identifier for the collection, which MUST match the specified pattern.
`title`	string A short descriptive one-line title for the collection.
`description` required	string <commonmark> Detailed multi-line description to explain the collection. CommonMark 0.29 syntax MAY be used for rich text representation.
`keywords`	Array of strings List of keywords describing the collection.
`version`	string Version of the collection. This property REQUIRES to add `version` (STAC < 1.0.0-rc.1) or `https://stac-extensions.github.io/version/v1.2.0/schema.json` (STAC >= 1.0.0-rc.1) to the list of `stac_extensions`.
`deprecated`	boolean Default: `false` Specifies that the collection is deprecated with the potential to be removed. It should be transitioned out of usage as soon as possible and users should refrain from using it in new projects. A link with relation type `latest-version` SHOULD be added to the links and MUST refer to the collection that can be used instead. This property REQUIRES to add `version` (STAC < 1.0.0-rc.1) or `https://stac-extensions.github.io/version/v1.2.0/schema.json` (STAC >= 1.0.0-rc.1) to the list of `stac_extensions`.
`license` required	string (stac_license) License(s) of the data as a SPDX License identifier. Alternatively, use `proprietary` if the license is not on the SPDX license list or `various` if multiple licenses apply. In these two cases links to the license texts SHOULD be added, see the `license` link relation type. Non-SPDX licenses SHOULD add a link to the license text with the `license` relation in the links section. The license text MUST NOT be provided as a value of this field. If there is no public license URL available, it is RECOMMENDED to host the license text and link to it.
`providers` >	Array of objects (stac_providers) A list of providers, which MAY include all organizations capturing or processing the data or the hosting provider. Providers SHOULD be listed in chronological order with the most recent provider being the last element of the list.
`extent` > required	object (Collection Extent) The extent of the data in the collection. Additional members MAY be added to represent other extents, for example, thermal or pressure ranges. The first item in the array always describes the overall extent of the data. All subsequent items describe more preciseextents, e.g. to identify clusters of data. Clients only interested in the overall extent will only need to access the first item in each array.
`links` > required	Array of objects (Link) Links related to this collection. Could reference to licensing information, other meta data formats with additional information or a preview image. Providing links with the following `rel` (relation) types is RECOMMENDED: 1. `root` and `parent`: URL to the data discovery endpoint at `/collections`. 2. `license`: A link to the license(s) SHOULD be specified if the `license` field is set to `proprietary` or `various`. 3. `example`: Links to examples of processes that use this collection. 4. `latest-version`: If a collection has been marked as deprecated, a link SHOULD point to the latest version of the collection. The relation types `predecessor-version` (link to older version) and `successor-version` (link to newer version) can also be used to show the relation between versions. 5. `alternate`: An alternative representation of the collection. For example, this could be the collection available through another catalog service such as OGC CSW, a human-readable HTML version or a metadata document following another standard such as ISO 19115 or DCAT. 6. `http://www.opengis.net/def/rel/ogc/1.0/queryables`: URL to the queryables endpoint at `/collections/{collection_id}/queryables`. For JSON Schema documents, the `type` field must be set to `application/schema+json`. For additional relation types see also the lists of common relation types in openEO and the STAC specification for Collections.
`cube:dimensions` > required	object (STAC Collection Cube Dimensions) The named default dimensions of the data cube. Names must be unique per collection. The keys of the object are the dimension names. For interoperability, it is RECOMMENDED to use the following dimension names if there is only a single dimension with the specified criteria: • `x` for the dimension of type `spatial` with the axis set to `x` • `y` for the dimension of type `spatial` with the axis set to `y` • `z` for the dimension of type `spatial` with the axis set to `z` • `t` for the dimension of type `temporal` • `bands` for dimensions of type `bands` • `geometry` for dimensions of type `geometry` This property REQUIRES to add a version of the data cube extension to the list of `stac_extensions`, e.g. `https://stac-extensions.github.io/datacube/v2.2.0/schema.json`.
`summaries` > required	object (STAC Summaries (Collection Properties)) Collection properties from STAC extensions (e.g. EO, SAR, Satellite or Scientific) or even custom extensions. Summaries are either a unique set of all available values, statistics *or* a JSON Schema. Statistics only specify the range (minimum and maximum values) by default, but can optionally be accompanied by additional statistical values. The range can specify the potential range of values, but it is recommended to be as precise as possible. The set of values MUST contain at least one element and it is strongly RECOMMENDED to list all values. It is recommended to list as many properties as reasonable so that consumers get a full overview of the Collection. Properties that are covered by the Collection specification (e.g. `providers` and `license`) SHOULD NOT be repeated in the summaries. Potential fields for the summaries can be found here: • **STAC Common Metadata**: A list of commonly used fields throughout all domains • **Content Extensions**: Domain-specific fields for domains such as EO, SAR and point clouds. • **Custom Properties**: It is generally allowed to add custom fields.
`assets` >	object (Assets) Dictionary of asset objects for data that can be downloaded, each with a unique key. The keys MAY be used by clients as file names. Implementing this property REQUIRES to add `collection-assets` to the list of `stac_extensions` in STAC < 1.0.0-rc.1.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "stac_version": "1.0.0",
  - "stac_extensions": [
        "https://stac-extensions.github.io/data
    ],
    "type": "Collection",
    "id": "Sentinel-2",
    "title": "Sentinel-2 MSI L2A",
    "description": "Sentinel-2A is a wide-swat
    "license": "proprietary",
  - "keywords": [
        "copernicus",
        "esa",
        "msi",
        "sentinel"
    ],
  - "providers": [
      + { … },
      + { … }
    ],
  - "extent": {
      + "spatial": { … },
      + "temporal": { … }
    },
  - "links": [
      + { … },
      + { … },
      + { … },
      + { … },
      + { … }
    ],
  - "cube:dimensions": {
      + "x": { … },
      + "y": { … },
      + "t": { … },
      + "bands": { … }
    },
  - "summaries": {
      + "constellation": [ … ],
      + "platform": [ … ],
      + "instruments": [ … ],
      + "eo:cloud_cover": { … },
      + "sat:orbit_state": [ … ],
      + "gsd": [ … ],
      + "eo:bands": [ … ],
      + "proj:epsg": { … }
    },
  - "assets": {
      + "thumbnail": { … },
      + "inspire": { … }
    }
}
```

Metadata filters for a specific dataset

GET `/collections/{collection_id}/queryables`

Lists **all** supported metadata filters (also called "queryables") for a specific collection.

This endpoint is compatible with the endpoint defined in the STAC API extension `filter` and OGC API - Features - Part 3: Filtering. For a precise definition please follow those specifications.

This endpoints provides a JSON Schema for each queryable that openEO users can use in multiple scenarios:

1. For loading data from the collection, e.g. in the process `load_collection`.
2. For filtering items using CQL2 on the `/collections/{collection_id}/items` endpoint (if STAC API - Features is implemented in addition to the openEO API).

Note: Providing the Bearer token is REQUIRED for private collections.

AUTHORIZATIONS: > None or *Bearer*

PATH PARAMETERS

`collection_id` required	string (collection_id) `^[\w\-\.~\/]+$` Example: `Sentinel-2A` Collection identifier

Responses

200 A JSON Schema defining the queryables. It is RECOMMENDED to dereference all "$refs".

RESPONSE SCHEMA: application/schema+json

`$schema`	string <uri> Default: `"http://json-schema.org/draft-07/schema#"` The JSON Schema version. If not given in the context of openEO, defaults to JSON Schema draft-07: `http://json-schema.org/draft-07/schema#` The default value for `$schema` property may have to be added to the JSON Schema object before passing it to a JSON Schema validator.
`$id`	string <uri> ID of your JSON Schema.
`type` >	json_schema_type (string) or Array of json_schema_type (strings) The allowed data type(s) for a value. If this property is not present, all data types are allowed.
`pattern`	string <regex> The regular expression a string value must match against.
`enum`	Array of any An exclusive list of allowed values.
`minimum`	number The minimum value (inclusive) allowed for a numerical value.
`maximum`	number The maximum value (inclusive) allowed for a numerical value.
`minItems`	number >= 0 Default: 0 The minimum number of items required in an array.
`maxItems`	number >= 0 The maximum number of items required in an array.
`items` >	Array of JSON Schema (objects) or JSON Schema (object) Specifies schemas for the items in an array.
`property name*` additional property	any Any other property supported by the JSON Schema version that is given through the property `$schema` are allowed. Defaults to JSON Schema draft-07, but can also be any later version of JSON Schema.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/schema+json

Copy Expand all Collapse all

```
{
    "$schema": "https://json-schema.org/draft/
    "$id": "https://openeo.example/api/v1/coll
    "type": "object",
    "title": "Sentinel-2A",
  - "properties": {
      + "eo:cloud_cover": { … },
      + "platform": { … }
    },
    "additionalProperties": false
}
```

Process Discovery

The process discovery endpoints provide details about the predefined processes that are available at the back-end. To list user-defined processes see 'User-Defined Processes'.

Supported predefined processes

GET `/processes`

Lists all predefined processes and returns detailed process descriptions, including parameters and return values.

AUTHORIZATIONS: > None or *Bearer*

QUERY PARAMETERS

`limit`	integer >= 1 Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 Formal specification describing the supported predefined processes.

RESPONSE SCHEMA: application/json

`processes` > required	Array of objects (Predefined Process)
`links` > required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "processes": [
      + { … },
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

List all user-defined processes

GET `/process_graphs`

Lists all user-defined processes (process graphs) of the authenticated user that are stored at the back-end.

It is **strongly RECOMMENDED** to keep the response size small by omitting larger optional values from the objects in `processes` (e.g. the `exceptions`, `examples` and `links` properties). To get the full metadata for a user-defined process clients MUST request `GET /process_graphs/{process_graph_id}`.

AUTHORIZATIONS: *Bearer*

QUERY PARAMETERS

`limit`	integer >= 1 Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 JSON array with user-defined processes.

RESPONSE SCHEMA: application/json

`processes` required	Array of objects (User-defined Process Metadata) Array of user-defined processes
`links` required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "processes": [
      + { … },
      + { … },
      + { … }
    ],
    "links": [ ]
}
```

Full metadata for a user-defined process

GET `/process_graphs/{process_graph_id}`

Lists all information about a user-defined process, including its process graph.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`process_graph_id` required	string (process_id) `^\w+$` Example: `ndvi` Per-user unique identifier for a user-defined process.

Responses

200 The user-defined process with process graph.

RESPONSE SCHEMA: application/json

`summary`	string or null A short summary of what the process does.
`description`	string or null <commonmark> Detailed description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation. In addition to the CommonMark syntax, clients can convert process IDs that are formatted as in the following example into links instead of code blocks: ``` ``process_id()`` ```
`parameters`	Array of objects or null (Process Parameter) A list of parameters. The order in the array corresponds to the parameter order to be used in clients that do not support named parameters. **Note:** Specifying an empty array is different from (if allowed) `null` or the property being absent. An empty array means the process has no parameters. `null` / property absent means that the parameters are unknown as the user has not specified them. There could still be parameters in the process graph, if one is specified.
`returns`	object or null (Process Return Value) Description of the data that is returned by this process.
`id` required	string (process_id) `^\w+$` The identifier for the process. It MUST be unique across its namespace (e.g. predefined processes or user-defined processes). Clients SHOULD warn the user if a user-defined process is added with the same identifier as one of the predefined process.
`categories`	Array of strings (process_categories) A list of categories.
`deprecated`	boolean (deprecated) Default: `false` Declares that the specified entity is deprecated with the potential to be removed in any of the next versions. It should be transitioned out of usage as soon as possible and users should refrain from using it in new implementations.
`experimental`	boolean (experimental) Default: `false` Declares that the specified entity is experimental, which means that it is likely to change or may produce unpredictable behavior. Users should refrain from using it in production, but still feel encouraged to try it out and give feedback.
`exceptions`	object (Process Exceptions) Declares exceptions (errors) that might occur during execution of this process. This list is just for informative purposes and may be incomplete. This list MUST only contain exceptions that stop the execution of a process and MUST NOT contain warnings, notices or debugging messages. It is meant to primarily contain errors that have been caused by the user. It is RECOMMENDED that exceptions are referred to and explained in process or parameter descriptions. The keys define the error code and MUST match the following pattern: `^\w+$` This schema follows the schema of the general openEO error list (see errors.json).
`examples`	Array of objects (Process Example) Examples, may be used for unit tests.
`links`	Array of objects (Link) Links related to this process, e.g. additional external documentation. Providing links with the following `rel` (relation) types is RECOMMENDED: 1. `latest-version`: If a process has been marked as deprecated, a link SHOULD point to the preferred version of the process. The relation types `predecessor-version` (link to older version) and `successor-version` (link to newer version) can also be used to show the relation between versions. 2. `example`: Links to examples of other processes that use this process. 3. `cite-as`: For all DOIs associated with the process, the respective DOI links SHOULD be added. For additional relation types see also the lists of common relation types in openEO.
`process_graph` required	object (Process Graph) A process graph defines a graph-like structure as a connected set of executable processes. Each key is a unique identifier (node ID) that is used to refer to the process in the graph.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

A user-defined process that computes the Enhanced Vegetation Index (EVI).

Copy Expand all Collapse all

```
{
    "id": "evi",
    "summary": "Enhanced Vegetation Index",
    "description": "Computes the Enhanced Vege
  - "parameters": [
      + { … },
      + { … },
      + { … }
    ],
  - "returns": {
        "description": "Computed EVI.",
      + "schema": { … }
    },
  - "process_graph": {
      + "sub": { … },
      + "p1": { … },
      + "p2": { … },
      + "sum": { … },
      + "div": { … },
      + "p3": { … }
    }
}
```

User-Defined Processes

These endpoints allow to store and manage user-defined processes with their process graphs at the back-end.

Validate a user-defined process (graph)

POST `/validation`

Validates a user-defined process without executing it. A user-defined process is considered valid unless the `errors` array in the response contains at least one error.

Checks whether the process graph is schematically correct and the processes are supported by the back-end. It MUST also check the arguments against the schema, but checking whether the arguments are adequate in the context of data is OPTIONAL. For example, a non-existing band name may get rejected only by a few back-ends. The validation MUST NOT throw an error for unresolvable process parameters.

Back-ends MUST validate the process graph. Validating the corresponding metadata is OPTIONAL.

Errors that usually occur during processing MAY NOT get reported, e.g. if a referenced file is accessible at the time of execution.

Back-ends can either report all errors at once or stop the validation once they found the first error.

Please note that a validation always returns with HTTP status code 200. Error codes in the 4xx and 5xx ranges MUST be returned only when the general validation request is invalid (e.g. server is busy or properties in the request body are missing), but never if an error was found during validation of the user-defined process (e.g. an unsupported process).

AUTHORIZATIONS: None or *Bearer*

REQUEST BODY SCHEMA: application/json
required

Specifies the user-defined process to be validated.

`id`	string or null `^\w+$` The identifier for the process. It MUST be unique across its namespace (e.g. predefined processes or user-defined processes). Clients SHOULD warn the user if a user-defined process is added with the same identifier as one of the predefined process.
`summary`	string or null A short summary of what the process does.
`description`	string or null <commonmark> Detailed description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation. In addition to the CommonMark syntax, clients can convert process IDs that are formatted as in the following example into links instead of code blocks: ``` ``process_id()`` ```
`parameters`	Array of objects or null (Process Parameter) A list of parameters. The order in the array corresponds to the parameter order to be used in clients that do not support named parameters. **Note:** Specifying an empty array is different from (if allowed) `null` or the property being absent. An empty array means the process has no parameters. `null` / property absent means that the parameters are unknown as the user has not specified them. There could still be parameters in the process graph, if one is specified.
`returns`	object or null (Process Return Value) Description of the data that is returned by this process.
`categories`	Array of strings (process_categories) A list of categories.
`deprecated`	boolean (deprecated) Default: `false` Declares that the specified entity is deprecated with the potential to be removed in any of the next versions. It should be transitioned out of usage as soon as possible and users should refrain from using it in new implementations.
`experimental`	boolean (experimental) Default: `false` Declares that the specified entity is experimental, which means that it is likely to change or may produce unpredictable behavior. Users should refrain from using it in production, but still feel encouraged to try it out and give feedback.
`exceptions`	object (Process Exceptions) Declares exceptions (errors) that might occur during execution of this process. This list is just for informative purposes and may be incomplete. This list MUST only contain exceptions that stop the execution of a process and MUST NOT contain warnings, notices or debugging messages. It is meant to primarily contain errors that have been caused by the user. It is RECOMMENDED that exceptions are referred to and explained in process or parameter descriptions. The keys define the error code and MUST match the following pattern: `^\w+$` This schema follows the schema of the general openEO error list (see errors.json).
`examples`	Array of objects (Process Example) Examples, may be used for unit tests.
`links`	Array of objects (Link) Links related to this process, e.g. additional external documentation. Providing links with the following `rel` (relation) types is RECOMMENDED: 1. `latest-version`: If a process has been marked as deprecated, a link SHOULD point to the preferred version of the process. The relation types `predecessor-version` (link to older version) and `successor-version` (link to newer version) can also be used to show the relation between versions. 2. `example`: Links to examples of other processes that use this process. 3. `cite-as`: For all DOIs associated with the process, the respective DOI links SHOULD be added. For additional relation types see also the lists of common relation types in openEO.
`process_graph` required	object (Process Graph) A process graph defines a graph-like structure as a connected set of executable processes. Each key is a unique identifier (node ID) that is used to refer to the process in the graph.

Responses

200 Returns the validation result as a list of errors. An empty list indicates a successful validation.

RESPONSE SCHEMA: application/json

`errors` required	Array of objects (General Error) A list of validation errors.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

A user-defined process that computes the Enhanced Vegetation Index (EVI).

Copy Expand all Collapse all

```
{
    "id": "evi",
    "summary": "Enhanced Vegetation Index",
    "description": "Computes the Enhanced Vege
  - "parameters": [
      + { … },
      + { … },
      + { … }
    ],
  - "returns": {
        "description": "Computed EVI.",
      + "schema": { … }
    },
  - "process_graph": {
      + "sub": { … },
      + "p1": { … },
      + "p2": { … },
      + "sum": { … },
      + "div": { … },
      + "p3": { … }
    }
}
```

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "errors": [
      + { … }
    ]
}
```

List all user-defined processes

GET `/process_graphs`

Lists all user-defined processes (process graphs) of the authenticated user that are stored at the back-end.

It is **strongly RECOMMENDED** to keep the response size small by omitting larger optional values from the objects in `processes` (e.g. the `exceptions`, `examples` and `links` properties). To get the full metadata for a user-defined process clients MUST request `GET /process_graphs/{process_graph_id}`.

AUTHORIZATIONS: *Bearer*

QUERY PARAMETERS

`limit`	integer >= 1 Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 JSON array with user-defined processes.

RESPONSE SCHEMA: application/json

`processes` required	Array of objects (User-defined Process Metadata) Array of user-defined processes
`links` required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "processes": [
      + { … },
      + { … },
      + { … }
    ],
    "links": [ ]
}
```

Full metadata for a user-defined process

GET `/process_graphs/{process_graph_id}`

Lists all information about a user-defined process, including its process graph.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`process_graph_id` required	string (process_id) `^\w+$` Example: `ndvi` Per-user unique identifier for a user-defined process.

Responses

200 The user-defined process with process graph.

RESPONSE SCHEMA: application/json

`summary`	string or null A short summary of what the process does.
`description`	string or null <commonmark> Detailed description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation. In addition to the CommonMark syntax, clients can convert process IDs that are formatted as in the following example into links instead of code blocks: ``` ``process_id()`` ```
`parameters`	Array of objects or null (Process Parameter) A list of parameters. The order in the array corresponds to the parameter order to be used in clients that do not support named parameters. **Note:** Specifying an empty array is different from (if allowed) `null` or the property being absent. An empty array means the process has no parameters. `null` / property absent means that the parameters are unknown as the user has not specified them. There could still be parameters in the process graph, if one is specified.
`returns`	object or null (Process Return Value) Description of the data that is returned by this process.
`id` required	string (process_id) `^\w+$` The identifier for the process. It MUST be unique across its namespace (e.g. predefined processes or user-defined processes). Clients SHOULD warn the user if a user-defined process is added with the same identifier as one of the predefined process.
`categories`	Array of strings (process_categories) A list of categories.
`deprecated`	boolean (deprecated) Default: `false` Declares that the specified entity is deprecated with the potential to be removed in any of the next versions. It should be transitioned out of usage as soon as possible and users should refrain from using it in new implementations.
`experimental`	boolean (experimental) Default: `false` Declares that the specified entity is experimental, which means that it is likely to change or may produce unpredictable behavior. Users should refrain from using it in production, but still feel encouraged to try it out and give feedback.
`exceptions`	object (Process Exceptions) Declares exceptions (errors) that might occur during execution of this process. This list is just for informative purposes and may be incomplete. This list MUST only contain exceptions that stop the execution of a process and MUST NOT contain warnings, notices or debugging messages. It is meant to primarily contain errors that have been caused by the user. It is RECOMMENDED that exceptions are referred to and explained in process or parameter descriptions. The keys define the error code and MUST match the following pattern: `^\w+$` This schema follows the schema of the general openEO error list (see errors.json).
`examples`	Array of objects (Process Example) Examples, may be used for unit tests.
`links`	Array of objects (Link) Links related to this process, e.g. additional external documentation. Providing links with the following `rel` (relation) types is RECOMMENDED: 1. `latest-version`: If a process has been marked as deprecated, a link SHOULD point to the preferred version of the process. The relation types `predecessor-version` (link to older version) and `successor-version` (link to newer version) can also be used to show the relation between versions. 2. `example`: Links to examples of other processes that use this process. 3. `cite-as`: For all DOIs associated with the process, the respective DOI links SHOULD be added. For additional relation types see also the lists of common relation types in openEO.
`process_graph` required	object (Process Graph) A process graph defines a graph-like structure as a connected set of executable processes. Each key is a unique identifier (node ID) that is used to refer to the process in the graph.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

A user-defined process that computes the Enhanced Vegetation Index (EVI).

Copy Expand all Collapse all

```
{
    "id": "evi",
    "summary": "Enhanced Vegetation Index",
    "description": "Computes the Enhanced Vege
  - "parameters": [
      + { … },
      + { … },
      + { … }
    ],
  - "returns": {
        "description": "Computed EVI.",
      + "schema": { … }
    },
  - "process_graph": {
      + "sub": { … },
      + "p1": { … },
      + "p2": { … },
      + "sum": { … },
      + "div": { … },
      + "p3": { … }
    }
}
```

Store a user-defined process

PUT `/process_graphs/{process_graph_id}`

Stores a provided user-defined process with process graph that can be reused in other processes.

If a process with the specified `process_graph_id` exists, the process is fully replaced. The id can not be changed for existing user-defined processes. The id MUST be unique across its namespace.

Partially updating user-defined processes is not supported.

To simplify exchanging user-defined processes, the property `id` can be part of the request body. If the values do not match, the value for `id` gets replaced with the value from the `process_graph_id` parameter in the path.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`process_graph_id` required	string (process_id) `^\w+$` Example: `ndvi` Per-user unique identifier for a user-defined process.

REQUEST BODY SCHEMA: application/json
required

Specifies the process graph with its meta data.

`id`	string or null `^\w+$` The identifier for the process. It MUST be unique across its namespace (e.g. predefined processes or user-defined processes). Clients SHOULD warn the user if a user-defined process is added with the same identifier as one of the predefined process.
`summary`	string or null A short summary of what the process does.

Request samples

Payload

Content type
application/json

A user-defined process that computes the Enhanced Vegetation Index (EVI).

Copy Expand all Collapse all

```
{
    "id": "evi",
    "summary": "Enhanced Vegetation Index",
    "description": "Computes the Enhanced Vege
  - "parameters": [
      + { … },
      + { … },
      + { … }
    ],
  - "returns": {
        "description": "Computed EVI.",
      + "schema": { … }
    },
  - "process_graph": {
      + "sub": { … },
      + "p1": { … },
      + "p2": { … },
      + "sum": { … },
      + "div": { … },
      + "p3": { … }
    }
}
```

`description`

string or null <commonmark>
Detailed description to explain the entity.

CommonMark 0.29 syntax MAY be used for rich text representation. In addition to the CommonMark syntax, clients can convert process IDs that are formatted as in the following example into links instead of code blocks: ``` ``process_id()`` ```

`parameters` ›

Array of objects or null (Process Parameter)
A list of parameters.

The order in the array corresponds to the parameter order to be used in clients that do not support named parameters.

Note: Specifying an empty array is different from (if allowed) `null` or the property being absent. An empty array means the process has no parameters. `null` / property absent means that the parameters are unknown as the user has not specified them. There could still be parameters in the process graph, if one is specified.

`returns` ›

object or null (Process Return Value)
Description of the data that is returned by this process.

`categories`

Array of strings (process_categories)
A list of categories.

`deprecated`

boolean (deprecated)
Default: `false`
Declares that the specified entity is deprecated with the potential to be removed in any of the next versions. It should be transitioned out of usage as soon as possible and users should refrain from using it in new implementations.

`experimental`

boolean (experimental)
Default: `false`
Declares that the specified entity is experimental, which means that it is likely to change or may produce unpredictable behavior. Users should refrain from using it in production, but still feel encouraged to try it out and give feedback.

`exceptions` ›

object (Process Exceptions)
Declares exceptions (errors) that might occur during execution of this process. This list is just for informative purposes and may be incomplete. This list MUST only contain exceptions that stop the execution of a process and MUST NOT contain warnings, notices or debugging messages. It is meant to primarily contain errors that have been caused by the user. It is RECOMMENDED that exceptions are referred to and explained in process or parameter descriptions.

The keys define the error code and MUST match the following pattern: `^\w+$`

This schema follows the schema of the general openEO error list (see errors.json).

`examples` ›

Array of objects (Process Example)
Examples, may be used for unit tests.

`links` ›

Array of objects (Link)
Links related to this process, e.g. additional external documentation.

Providing links with the following `rel` (relation) types is RECOMMENDED:

1. `latest-version`: If a process has been marked as deprecated, a link SHOULD point to the preferred version of the process. The relation types `predecessor-version` (link to older version) and `successor-version` (link to newer version) can also be used to show the relation between versions.
2. `example`: Links to examples of other processes that use this process.
3. `cite-as`: For all DOIs associated with the process, the respective DOI links SHOULD be added.

For additional relation types see also the lists of common relation types in openEO.

`process_graph` › required

object (Process Graph)
A process graph defines a graph-like structure as a connected set of executable processes. Each key is a unique identifier (node ID) that is used to refer to the process in the graph.

Responses

— **200** The user-defined process has been stored successfully.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Delete a user-defined process

DELETE `/process_graphs/{process_graph_id}`

Deletes the data related to this user-defined process, including its process graph.

Does NOT delete jobs or services that reference this user-defined process.

AUTHORIZATIONS: › *Bearer*

PATH PARAMETERS

`process_graph_id` required

string (process_id) `^\w+$`
Example: `ndvi`
Per-user unique identifier for a user-defined process.

Responses

— **204** The user-defined process has been successfully deleted

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Data Processing

Organizes and manages data processing on the back-end, either as synchronous on-demand computation or batch jobs.

Supported file formats

GET `/file_formats`

Lists supported input and output file formats. *Input* file formats specify which file a back-end can *read* from. *Output* file formats specify which file a back-end can *write* to.

The response to this request is an object listing all available input and output file formats separately with their parameters and additional data. This endpoint does not include the supported secondary web services.

Note: Format names and parameters MUST be fully aligned with the GDAL codes if available, see GDAL Raster Formats and OGR Vector Formats. It is OPTIONAL to support all output format parameters supported by GDAL. Some file formats not available through GDAL may be defined centrally for openEO. Custom file formats or parameters MAY be defined.

The format descriptions MUST describe how the file formats relate to data cubes. Input file formats MUST describe how the files have to be structured to be transformed into data cubes. Output file formats MUST describe how the data cubes are stored at the back-end and how the resulting file structure looks like.

Back-ends MUST NOT support aliases, for example it is not allowed to support `geotiff` instead of `gtiff`. Nevertheless, openEO Clients MAY translate user input for convenience (e.g. translate `geotiff` to `gtiff`). Also, for a better user experience the back-end can specify a `title`.

Format names MUST be accepted in a *case insensitive* manner throughout the API.

AUTHORIZATIONS: › None or *Bearer*

Responses

⌄ **200**
An object with containing all input and output format separately. For each property `input` and `output` an object is defined where the file format names are the property keys and the property values are objects that define a title, supported parameters and related links.

RESPONSE SCHEMA: application/json

`input` › required

object (Input File Formats)
Map of supported input file formats, i.e. file formats a back-end can **read** from. The property keys are the file format names that are used by clients and users, for example in process graphs.

`output` › required

object (Output File Formats)
Map of supported output file formats, i.e. file formats a back-end can **write** to. The property keys are the file format names that are used by clients and users, for example in process graphs.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request usually does not respond with HTTP status codes 401 and 403 due to missing authorization. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "output": {
      + "GTiff": { … },
      + "GPKG": { … }
    },
  - "input": {
      + "GPKG": { … }
    }
}
```

Validate a user-defined process (graph)

POST `/validation`

Validates a user-defined process without executing it. A user-defined process is considered valid unless the `errors` array in the response contains at least one error.

Checks whether the process graph is schematically correct and the processes are supported by the back-end. It MUST also check the arguments against the schema, but checking whether the arguments are adequate in the context of data is OPTIONAL. For example, a non-existing band name may get rejected only by a few back-ends. The validation MUST NOT throw an error for unresolvable process parameters.

Back-ends MUST validate the process graph. Validating the corresponding metadata is OPTIONAL.

Errors that usually occur during processing MAY NOT get reported, e.g. if a referenced file is accessible at the time of execution.

Back-ends can either report all errors at once or stop the validation once they found the first error.

Please note that a validation always returns with HTTP status code 200. Error codes in the 4xx and 5xx ranges MUST be returned only when the general validation request is invalid (e.g. server is busy or properties in the request body are missing), but never if an error was found during validation of the user-defined process (e.g. an unsupported process).

AUTHORIZATIONS: › None or *Bearer*

REQUEST BODY SCHEMA: application/json
required

Specifies the user-defined process to be validated.

`id`

string or null `^\w+$`
The identifier for the process. It MUST be unique across its namespace (e.g. predefined processes or user-defined processes).

Clients SHOULD warn the user if a user-defined process is added with the same identifier as one of the predefined process.

`summary`

string or null
A short summary of what the process does.

`description`

string or null <commonmark>
Detailed description to explain the entity.

CommonMark 0.29 syntax MAY be used for rich text representation. In addition to the CommonMark syntax, clients can convert process IDs that are formatted as in the following example into links instead of code blocks: ``` ``process_id()`` ```

`parameters` ›

Array of objects or null (Process Parameter)
A list of parameters.

The order in the array corresponds to the parameter order to be used in clients that do not support named parameters.

Note: Specifying an empty array is different from (if allowed) `null` or the property being absent. An empty array means the process has no parameters. `null` / property absent means that the parameters are unknown as the user has not specified them. There could still be parameters in the process graph, if one is specified.

`returns` ›

object or null (Process Return Value)
Description of the data that is returned by this process.

`categories`

Array of strings (process_categories)
A list of categories.

`deprecated`

boolean (deprecated)
Default: `false`
Declares that the specified entity is deprecated with the potential to be removed in any of the next versions. It should be transitioned out of usage as soon as possible and users should refrain from using it in new implementations.

`experimental`

boolean (experimental)
Default: `false`
Declares that the specified entity is experimental, which means that it is likely to change or may produce unpredictable behavior. Users should refrain from using it in production, but still feel encouraged to try it out and give feedback.

`exceptions` ›

object (Process Exceptions)
Declares exceptions (errors) that might occur during execution of this process. This list is just for informative purposes and may be incomplete. This list MUST only contain exceptions that stop the execution of a process and MUST NOT contain warnings, notices or debugging messages. It is meant to primarily contain errors that have been caused by the user. It is RECOMMENDED that exceptions are referred to and explained in process or parameter descriptions.

The keys define the error code and MUST match the following pattern: `^\w+$`

This schema follows the schema of the general openEO error list (see errors.json).

`examples` ›

Array of objects (Process Example)
Examples, may be used for unit tests.

`links` ›

Array of objects (Link)
Links related to this process, e.g. additional external documentation.

Providing links with the following `rel` (relation) types is RECOMMENDED:

1. `latest-version`: If a process has been marked as deprecated, a link SHOULD point to the preferred version of the process. The relation types `predecessor-version` (link to older version) and `successor-version` (link to newer version) can also be used to show the relation between versions.
2. `example`: Links to examples of other processes that use this process.
3. `cite-as`: For all DOIs associated with the process, the respective DOI links SHOULD be added.

For additional relation types see also the lists of common relation types in openEO.

`process_graph` › required

object (Process Graph)
A process graph defines a graph-like structure as a connected set of executable processes. Each key is a unique identifier (node ID) that is used to refer to the process in the graph.

Responses

⌄ **200** Returns the validation result as a list of errors. An empty list indicates a successful validation.

RESPONSE SCHEMA: application/json

`errors` › required

Array of objects (General Error)
A list of validation errors.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

A user-defined process that computes the Enhanced Vegetation Index (EVI).

Copy Expand all Collapse all

```
{
    "id": "evi",
    "summary": "Enhanced Vegetation Index",
    "description": "Computes the Enhanced Vege
  - "parameters": [
      + { … },
      + { … },
      + { … }
    ],
  - "returns": {
        "description": "Computed EVI.",
      + "schema": { … }
    },
  - "process_graph": {
      + "sub": { … },
      + "p1": { … },
      + "p2": { … },
      + "sum": { … },
      + "div": { … },
      + "p3": { … }
    }
}
```

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "errors": [
      + { … }
    ]
}
```

Process and download data synchronously

POST `/result`

Executes a user-defined process directly (synchronously) and the result will be downloaded in the format specified in the process graph. This endpoint can be used to generate small previews or test user-defined processes before starting a batch job. Timeouts on either client-side or server-side are to be expected for complex computations. Back-ends MAY send the openEO error `ProcessGraphComplexity` immediately if the computation is expected to time out. Otherwise requests MAY time-out after a certain amount of time by sending openEO error `RequestTimeout`. A header named `OpenEO-Costs` MAY be sent with all responses, which MUST include the costs for processing and downloading the data. Additionally, a link to a log file MAY be sent in the header.

AUTHORIZATIONS: › *Bearer*

REQUEST BODY SCHEMA: application/json
required

Specifies the job details, e.g. the user-defined process and billing details.

`process` › required

object (Process Graph with metadata)
A process graph, optionally enriched with process metadata.

`budget`

number or null (budget) >= 0
Default: `null`
Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.

`plan`

string or null (billing_plan_null_default)
Default: `null`
The billing plan to process and charge the job or service with.

Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way:

- If a non-`null` value is given: Persist the `plan` that has been provided in the request.
- Otherwise:
  1. Persist the `default_plan` exposed through `GET /me`, if available.
  2. Persist the `default_plan` exposed through `GET /`, if available.
  3. If a single plan is exposed by the back-end, persist it.
  4. Otherwise, the back-end MUST throw a `BillingPlanMissing` error.

The resolved plan MUST be persisted permanently, regardless of any changes to the exposed billing plans in `GET /` in the future.

Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.

`log_level`

string (min_log_level_default)
Default: `"info"`
Enum: `"error"` `"warning"` `"info"` `"debug"`
The minimum severity level for log entries that the back-end stores for the processing request.

The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`.

The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.

`property name*` additional property

any
Aditional back-end specific properties are allowed.

Responses

⌄ **200** Result data in the requested output format

RESPONSE HEADERS

`Content-Type`

string
The appropriate media (MIME) type for the requested output format MUST be sent, if the response contains a single file.

To send multiple files at once it is RECOMMENDED to use the `tar` file format (media type: `application/x-tar`).

To mimic the results of batch jobs, it is RECOMMENDED that

1. clients extract the tar file directly after receiving it so that users can directly work on the contained files *and*
2. back-ends add STAC Items and/or Collections to the tar file so that users can make sense of the files.

`OpenEO-Costs`

number or null (money) >= 0
Default: `null`
Example: `"12.98"`
MAY include the costs for processing and downloading the data.

`Link`

string `^<[^>]+>;\s?rel="monitor"`
Example: `"<https://openeo.example/api/v1/logs/258489231>; rel=\"monitor\""`
The header MAY indicate a link to a log file generated by the request. If provided, the link MUST be serialized according to RFC 8288 and MUST use the relation type `monitor`. The link MUST follow the specifications for the links `GET /jobs/{job_id}/logs` and `GET /services/{service_id}/logs`, except that is MUST NOT accept any parameters (limit/offset). Therefore, the link MUST be accessible with HTTP GET, MUST be secured using a Bearer token and MUST follow the corresponding request body schema.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
    "budget": 100,
    "plan": "free",
    "log_level": "warning",
    "property1": null,
    "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

List all batch jobs

GET `/jobs`

Lists all batch jobs submitted by a user.

It is **strongly RECOMMENDED** to keep the response size small by omitting all optional non-scalar values (i.e. arrays and objects) from objects in `jobs` (i.e. the `process` property). To get the full metadata for a job clients MUST request `GET /jobs/{job_id}`.

AUTHORIZATIONS: › *Bearer*

QUERY PARAMETERS

`limit`

integer >= 1
Example: `limit=10`
This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned.

Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources.

If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

⌄ **200** Array of job descriptions

RESPONSE SCHEMA: application/json

`jobs` › required

Array of objects (Batch Job)

`links` › required

Array of objects (links_pagination)
Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated.

If pagination is implemented, the following `rel` (relation) types apply:

1. `next` (REQUIRED): A link to the next page, except on the last page.
2. `prev` (OPTIONAL): A link to the previous page, except on the first page.
3. `first` (OPTIONAL): A link to the first page, except on the first page.
4. `last` (OPTIONAL): A link to the last page, except on the last page.

For additional relation types see also the lists of common relation types in openEO.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "jobs": [
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

Create a new batch job

POST `/jobs`

Creates a new batch processing task (job) from one or more (chained) processes at the back-end.

Processing the data does not start yet. The job status gets initialized as `created` by default.

AUTHORIZATIONS: › *Bearer*

REQUEST BODY SCHEMA: application/json
required

Specifies the job details, e.g. the user-defined process and billing details.

`title`

string or null (eo_title)
A short description to easily distinguish entities.

`description`

string or null <commonmark> (eo_description)
Detailed multi-line description to explain the entity.

CommonMark 0.29 syntax MAY be used for rich text representation.

`process` › required

object (Process Graph with metadata)
A process graph, optionally enriched with process metadata.

`plan`

string or null (billing_plan_null_default)
Default: `null`
The billing plan to process and charge the job or service with.

Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way:

- If a non-`null` value is given: Persist the `plan` that has been provided in the request.
- Otherwise:
  1. Persist the `default_plan` exposed through `GET /me`, if available.
  2. Persist the `default_plan` exposed through `GET /`, if available.
  3. If a single plan is exposed by the back-end, persist it.
  4. Otherwise, the back-end MUST throw a `BillingPlanMissing` error.

The resolved plan MUST be persisted permanently, regardless of any changes to the exposed billing plans in `GET /` in the future.

Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.

`budget`

number or null (budget) >= 0
Default: `null`
Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.

`log_level`

string (min_log_level_default)
Default: `"info"`
Enum: `"error"` `"warning"` `"info"` `"debug"`
The minimum severity level for log entries that the back-end stores for the processing request.

The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`.

The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.

`property name*` additional property

any
Additional back-end specific properties are allowed.

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
    "title": "NDVI based on Sentinel-2",
    "description": "Deriving minimum NDVI meas
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
    "plan": "free",
    "budget": 100,
    "log_level": "warning",
    "property1": null,
    "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Responses

201 The batch job has been created successfully.

RESPONSE HEADERS

`Location` required	string <uri> Example: `"https://openeo.example/api/v1/jobs/123"` Absolute URL to the newly created batch job. The URL points to the metadata endpoint `GET /jobs/{job_id}` with the `{job_id}` being the unique identifier (ID) of the created batch job.
`OpenEO-Identifier` required	string (job_id) `^[\w\-\.~]+$` Example: `"a3cca2b2aa1e3b5b"` Per-back-end unique identifier of the batch job, generated by the back-end during creation. MUST match the specified pattern.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Modify a batch job

PATCH `/jobs/{job_id}`

Modifies an existing job at the back-end, but maintains the identifier. Changes can be grouped in a single request.

The job status does not change.

Jobs can only be modified when the job is not queued and not running. Otherwise, requests to this endpoint MUST be rejected with openEO error `JobLocked`.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required	string (job_id) `^[\w\-\.~]+$` Example: `a3cca2b2aa1e3b5b` Identifier of the batch job.

REQUEST BODY SCHEMA: application/json
required

Specifies the job details to update.

`title`	string or null (eo_title) A short description to easily distinguish entities.
`description`	string or null <commonmark> (eo_description) Detailed multi-line description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation.
`process` ›	object (Process Graph with metadata) A process graph, optionally enriched with process metadata.
`plan`	string or null (billing_plan_null) The billing plan to process and charge the job or service with. Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way if billing is supported: • If a value is given and it is not `null`: Persist the `plan` that has been provided in the request. • Otherwise, do not change the billing plan. Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.
`budget`	number or null (budget_update) `>= 0` Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null`.
`log_level`	string (min_log_level_update) Enum: `"error"` `"warning"` `"info"` `"debug"` Updates the minimum severity level for log entries that the back-end stores for the processing requests. The back-end does not need to update existing log entries.
`property name*` additional property	any Additional back-end specific properties are allowed.

Responses

204 Changes to the job applied successfully.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
    "title": "NDVI based on Sentinel-2",
    "description": "Deriving minimum NDVI meas
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
    "plan": "free",
    "budget": 100,
    "log_level": "warning",
    "property1": null,
    "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Full metadata for a batch job

GET `/jobs/{job_id}`

Lists all information about a submitted batch job.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required	string (job_id) `^[\w\-\.~]+$` Example: `a3cca2b2aa1e3b5b` Identifier of the batch job.

Responses

200 Full job information.

RESPONSE SCHEMA: application/json

`id` required	string (job_id) `^[\w\-\.~]+$` Per-back-end unique identifier of the batch job, generated by the back-end during creation. MUST match the specified pattern.
`title`	string or null (eo_title) A short description to easily distinguish entities.
`description`	string or null <commonmark> (eo_description) Detailed multi-line description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation.
`process` › required	object (Process Graph with metadata) A process graph, optionally enriched with process metadata.

`status` required	string
	Default: `"created"`
	Enum: `"created"` `"queued"` `"running"` `"canceled"` `"finished"` `"error"`
	The current status of a batch job.
	The following status changes can occur:
	• `POST /jobs`: The status is initialized as `created`.
	• `POST /jobs/{job_id}/results`: The status is set to `queued`, if processing does not start instantly.
	  ◦ Once the processing starts the status is set to `running`.
	  ◦ Once the data is available to download the status is set to `finished`.
	  ◦ Whenever an error occurs during processing, the status MUST be set to `error`.
	• `DELETE /jobs/{job_id}/results`: The status is set to `canceled` if the status was `running` beforehand and partial or preliminary results are available to be downloaded. Otherwise the status is set to `created`.

`progress`	number `[ 0 .. 100 ]` Indicates the process of a running batch job, in percent. Can also be set for a job which stopped due to an error or was canceled by the user. In this case, the value indicates the progress at which the job stopped. This property may not be available for the status codes `created` and `queued`. Submitted and queued jobs only allow the value `0`, finished jobs only allow the value `100`.
`created` required	string <date-time> (created) Date and time of creation, formatted as a RFC 3339 date-time.
`updated`	string <date-time> (updated) Date and time of the last status change, formatted as a RFC 3339 date-time.
`plan`	string (billing_plan) The billing plan to process and charge the job or service with. Billing plans MUST be handled in a *case insensitive* manner. The plans can be retrieved from `GET /`, but the value returned here may not be in the list of plans any longer.
`costs`	number or null (money) `>= 0` Default: `null` An amount of money or credits. The value MUST be specified in the currency the back-end is working with. The currency can be retrieved by calling `GET /`. If no currency is set, this field MUST be `null`.
`budget`	number or null (budget) `>= 0` Default: `null` Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.
`usage` ›	object (Resource usage metrics) Metrics about the resource usage of the batch job. Back-ends are not expected to update the metrics while processing data, so the metrics can only be available after the job has finished or has stopped due to an error. For usage metrics during processing, metrics can better be added to the logs (e.g. `GET /jobs/{job_id}/logs`) with the same schema.
`log_level`	string (min_log_level_default) Default: `"info"` Enum: `"error"` `"warning"` `"info"` `"debug"` The minimum severity level for log entries that the back-end stores for the processing request. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`. The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.
`links` ›	Array of objects (Link) Links related to this batch job such as links to invoices, log files or results. Providing links with the following `rel` (relation) types is RECOMMENDED: 1. `monitor`: If logs are available, a link to the logs endpoint. 2. `result`: If batch job results are available, a link to the results endpoint. The relation types `monitor` and `result` may occur for various batch job states: 1. `created`: When the batch job was executed before and has been reset to `created` after an update there could still be results and logs available until they get discarded by queueing the batch job again. 2. `finished`: The full log and results are expected to be available. 3. `error` / `canceled`: Partial results and logs may be available. For more relation types see the lists of common relation types in openEO.
`property name*` additional property	any You can list additional back-end specific properties here.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "property1": null,
    "property2": null,
    "id": "a3cca2b2aa1e3b5b",
    "title": "NDVI based on Sentinel-2",
    "description": "Deriving minimum NDVI meas
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
    "status": "running",
    "progress": 75.5,
    "created": "2017-01-01T09:32:12Z",
    "updated": "2017-01-01T09:36:18Z",
    "plan": "free",
    "costs": 12.98,
    "budget": 100,
  - "usage": {
      + "cpu": { … },
      + "duration": { … },
      + "memory": { … },
      + "network": { … },
      + "storage": { … }
    },
    "log_level": "warning",
  - "links": [
      + { … },
      + { … }
    ]
}
```

Delete a batch job

DELETE `/jobs/{job_id}`

Deletes all data related to a given batch job. Computations are stopped and computed results are deleted. This job will not generate additional costs for processing.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required	string (job_id) `^[\w\-\.~]+$` Example: `a3cca2b2aa1e3b5b` Identifier of the batch job.

Responses

204 The job has been successfully deleted.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Get an estimate for a batch job

GET `/jobs/{job_id}/estimate`

Calculates an estimate for a batch job. Back-ends can decide to either calculate the duration, the costs, the size or a combination of them. Back-end providers MAY specify an expiry time for the estimate. Starting to process data afterwards MAY be charged at a higher cost. Costs do often not include download costs. Whether download costs are included or not can be indicated explicitly with the `downloads_included` flag. The estimate SHOULD be the upper limit of the costs, but back-end are free to use the field according to their terms of service. For some batch jobs it is not (easily) possible to estimate the costs reliably, e.g. if a UDF or ML model is part of the process. In this case, the server SHOULD return a `EstimateComplexity` error with HTTP status code 500.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required	string (job_id) `^[\w\-\.~]+$` Example: `a3cca2b2aa1e3b5b` Identifier of the batch job.

Responses

200
The estimated costs with regard to money, processing time and storage capacity. At least one of `costs`, `duration` or `size` MUST be provided.

RESPONSE SCHEMA: application/json

Any of **Batch Job Estimate** Batch Job Estimate Batch Job Estimate

`costs` required	number or null (money) `>= 0` Default: `null` An amount of money or credits. The value MUST be specified in the currency the back-end is working with. The currency can be retrieved by calling `GET /`. If no currency is set, this field MUST be `null`.
`duration`	string Estimated duration for the operation. Duration MUST be specified as an ISO 8601 duration.
`size`	integer Estimated required storage capacity, i.e. the size of the generated files. Size MUST be specified in bytes.
`downloads_included`	integer or null Default: `null` Specifies how many full downloads of the processed data are included in the estimate. Set to `null` for unlimited downloads, which is also the default value.
`expires`	string <date-time> Time until which the estimate is valid, formatted as a RFC 3339 date-time.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy

```
{
    "costs": 12.98,
    "duration": "P1Y2M10DT2H30M",
    "size": 157286400,
    "downloads_included": 5,
    "expires": "2020-11-01T00:00:00Z"
}
```

Logs for a batch job

GET `/jobs/{job_id}/logs`

Lists log entries for the batch job, usually for debugging purposes.

Back-ends can log any information that may be relevant for a user at any stage (status) of the batch job. Users can log information during data processing using respective processes such as `inspect`.

If requested consecutively, it is RECOMMENDED that clients use the offset parameter to get only the entries they have not received yet.

While pagination itself is OPTIONAL, the `offset` parameter is REQUIRED to be implemented by back-ends.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required	string (job_id) `^[\w\-\.~]+$` Example: `a3cca2b2aa1e3b5b` Identifier of the batch job.

QUERY PARAMETERS

`offset`	string Example: `offset=log1234` The last identifier (property `id` of a log entry) the client has received. If provided, the back-end MUST only send the entries that occurred after the specified identifier. If not provided or empty, the back-end MUST start with the first entry.
`level`	string Default: `"debug"` Enum: `"error"` `"warning"` `"info"` `"debug"` Example: `level=error` The minimum severity level for log entries that the back-end returns. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only return log entries with the level `warning` and `error`. The default minimum log level is `debug`, which returns all log levels.
`limit`	integer `>= 1` Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 Lists the requested log entries.

RESPONSE SCHEMA: application/json

`level`	string Default: `"debug"` Enum: `"error"` `"warning"` `"info"` `"debug"` The minimum severity level for log entries that the back-end returns. This property MUST reflect the effective lowest `level` that may appear in the document, which is (if implemented) the highest level of: 1. the `log_level` specified by the user for the processing request. 2. the `level` specified by the user for the log request. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the logs will only contain entries with the level `warning` and `error`.
`logs` › required	Array of objects (Log Entry) A chronological list of logs.
`links` › required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "level": "error",
  - "logs": [
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

List batch job results

GET `/jobs/{job_id}/results`

Lists signed URLs pointing to the processed files, usually after the batch job has finished. Back-ends may also point to intermediate results after the job has stopped due to an error or if the `partial` parameter has been set.

The response includes additional metadata. It is a valid STAC Item (if it has spatial and temporal references included) or a valid STAC Collection (supported since openEO API version 1.1.0). The assets to download are in both cases available in the property `assets` and have the same structure. Additional metadata is not strictly required to download the files, but are helpful for users to understand the data.

STAC Collections can either (1) add all assets as collection-level assets or (2) link to STAC Catalogs and STAC Items with signed URLs, which will provide a full STAC catalog structure a client has to go through. Option 2 is overall the better architectural choice and allows a fine-grained description of the processed data, but it is not compliant with previous versions of the openEO API. **To maintain backward compatibility, it is REQUIRED to still copy all assets in the STAC catalog structure into the collection-level assets.** This requirement is planned to be removed in openEO API version 2.0.0. A client can enforce that the server returns a GeoJSON through content negotiation with the media type `application/geo+json`, but the results may not contain very meaningful metadata aside from the assets.

Clients are RECOMMENDED to store this response and all potential sub-catalogs and items with the assets so that the downloaded data is then a self-contained STAC catalog user could publish easily with all the data and metadata.

URL signing is a way to protect files from unauthorized access with a key in the URL instead of HTTP header based authorization. The URL signing key is similar to a password and its inclusion in the URL allows to download files using simple GET requests supported by a wide range of programs, e.g. web browsers or download managers. Back-ends are responsible to generate the URL signing keys and to manage their appropriate expiration. The back-end MAY indicate an expiration time by setting the `expires` property in the reponse. Requesting this endpoint SHOULD always return non-expired URLs. Signed URLs that were generated for a previous request and already expired SHOULD NOT be reused, but regenerated with new expiration time. Signed URLs that expired MAY return the openEO error `ResultLinkExpired`.

Adding a link with relation type `canonical` to the STAC Item or STAC Collection is STRONGLY RECOMMENDED (see the `links` property for details).

If processing has not finished yet and the `partial` parameter is not set to `true` requests to this endpoint MUST be rejected with openEO error `JobNotFinished`.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required	string (job_id) `^[\w\-\.~]+$` Example: `a3cca2b2aa1e3b5b` Identifier of the batch job.

QUERY PARAMETERS

`partial`	boolean Default: `false` If set to `true`, the results endpoint returns incomplete results while still running. Enabling this parameter requires to indicate the status of the batch job in the STAC metadata by setting the `openeo:status`.

Responses

200
Valid download links have been returned. The download links does not necessarily need to be located under the API base url.

RESPONSE HEADERS

`OpenEO-Costs`	number or null (money) `>= 0` Default: `null` Example: `"12.98"` Specifies the costs for fully downloading the data **once**, i.e. this header MAY change in subsequent calls. It MUST be set to `0` if the requester is the owner of the job and still has free downloads included in his processing charges estimated by `GET /jobs/{job_id}/estimate`. If a requester other than the owner is requesting the data of a shared job this header indicates the costs for the requester.

RESPONSE SCHEMA: application/json

One of **Batch Job Results Response as STAC Item** Batch Job Results Response as STAC Collection

`stac_version` required	string (stac_version) `^(0\.9.\d+|1\.\d+.\d+)` The version of the STAC specification, which MAY not be equal to the STAC API version. Supports versions 0.9.x and 1.x.x.
`stac_extensions` ›	Array of Reference to a JSON Schema (string) or Reference to a core extension (STAC < 1.0.0-rc.1 only) (string) (stac_extensions) `unique` A list of implemented STAC extensions. The list contains URLs to the JSON Schema files it can be validated against. For STAC < 1.0.0-rc.1 shortcuts such as `sar` can be used instead of the schema URL.
`id` required	string (job_id) `^[\w\-\.~]+$` Per-back-end unique identifier of the batch job, generated by the back-end during creation. MUST match the specified pattern.
`type` required	string Value: `"Feature"` The GeoJSON type that applies to this metadata document, which MUST always be a "Feature" according to the STAC specification. This type does **not** describe the spatial data type of the assets.
`bbox` ›	Array of 4 elements (numbers) or Array of 6 elements (numbers) (bbox) Each bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth): • West (lower left corner, coordinate axis 1) • South (lower left corner, coordinate axis 2) • Base (optional, minimum value, coordinate axis 3) • East (upper right corner, coordinate axis 1) • North (upper right corner, coordinate axis 2) • Height (optional, maximum value, coordinate axis 3) The coordinate reference system of the values is WGS 84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84). For WGS 84 longitude/latitude the values are in most cases the

Response samples

200 424 4XX 5XX

Content type
application/json

Example
Batch Job Results Response as STAC Item

Copy Expand all Collapse all

```
{
    "stac_version": "1.0.0",
  - "stac_extensions": [
        "https://openeo.example/stac/custom-ext
    ],
    "id": "a3cca2b2aa1e3b5b",
    "type": "Feature",
  - "bbox": [
        -180,
        -90,
        180,
        90
    ],
  - "geometry": {
        "type": "Polygon",
      + "coordinates": [ … ]
    },
  - "properties": {
        "datetime": "2019-08-24T14:15:22Z",
        "start_datetime": "2019-08-24T14:15:22
        "end_datetime": "2019-08-24T14:15:22Z"
        "title": "NDVI based on Sentinel-2",
        "description": "Deriving minimum NDVI m
        "license": "Apache-2.0",
      + "providers": [ … ],
        "created": "2017-01-01T09:32:12Z",
        "updated": "2017-01-01T09:36:18Z",
        "expires": "2020-11-01T00:00:00Z",
        "openeo:status": "running"
    },
  - "assets": {
      + "preview.png": { … },
      + "process.json": { … },
      + "1.tif": { … },
      + "2.tif": { … },
      + "inspire.xml": { … }
    },
  - "links": [
      + { … }
    ]
}
```

For WGS 84 longitude/latitude the values are in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude.

However, in cases where the box spans the antimeridian the first value (west-most box edge) is larger than the third value (east-most box edge).

If the vertical axis is included, the third and the sixth number are the bottom and the top of the 3-dimensional bounding box.

geometry › `required`
object or null (GeoJSON Geometry)
Defines the full footprint of the asset represented by this item as GeoJSON Geometry.

Results without a known location can set this value to `null`.

properties › `required`
object (Item Properties)
MAY contain additional properties other than the required property `datetime`, e.g. custom properties or properties from the STAC specification or STAC extensions.

assets › `required`
object (Assets)
Dictionary of asset objects for data that can be downloaded, each with a unique key. The keys MAY be used by clients as file names.

links › `required`
Array of objects (Link)
Links related to this batch job result, e.g. a link to an invoice, additional log files or external documentation.

The links MUST NOT contain links to the processed and downloadable data. Instead specify these in the `assets` property. Clients MUST NOT download the data referenced in the links by default.

It is **strongly recommended** to add a link with relation type `canonical`, which points to this STAC document using a signed URL. This way the STAC metadata can be used by non-openEO clients without additional authentication steps.

For relation types see the lists of common relation types in openEO.

› **424**
The request can not be fulfilled as the batch job failed. This request will deliver the last error message that was produced by the batch job.

This HTTP code MUST be sent only when the job `status` is `error`.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Start processing a batch job

POST `/jobs/{job_id}/results`

Adds a batch job to the processing queue to compute the results.

The result will be stored in the format specified in the process. To specify the format use a process such as `save_result`.

The job status is set to `queued`, if processing does not start instantly. The same applies if the job status is `canceled`, `finished`, or `error`, which restarts the job and discards previous results if the back-end does not reject the request with an error. Clients SHOULD warn users and ask for confirmation if results may get discarded.

- Once the processing starts the status is set to `running`.
- Once the data is available to download the status is set to `finished`.
- Whenever an error occurs during processing, the status MUST be set to `error`.

This endpoint has no effect if the job status is already `queued` or `running`. In particular, it does not restart a running job. To restart a queued or running job, processing MUST have been canceled.

Back-ends SHOULD reject queueing jobs with openEO error `PaymentRequired`, if the back-end is able to detect that the budget is too low to fully process the request. Alternatively, back-ends MAY provide partial results once reaching the budget. If none of the alternatives is feasible, the results are discarded. Thus, client SHOULD warn users that reaching the budget may lead to partial or no results at all.

AUTHORIZATIONS: › *Bearer*

PATH PARAMETERS

job_id `required`
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

— **202** The creation of the resource has been queued successfully.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  "id": "550e8400-e29b-11d4-a716-44665544000
  "code": "SampleError",
  "message": "Parameter 'sample' is missing.
- "links": [
    + { … }
  ]
}
```

Cancel processing a batch job

DELETE `/jobs/{job_id}/results`

Cancels all related computations for this job at the back-end. It will stop generating additional costs for processing.

A subset of processed results may be available for downloading depending on the state of the job at the time it was canceled.

Results MUST NOT be deleted until the job processing is started again or the job is completely deleted through a request to `DELETE /jobs/{job_id}`.

This endpoint only has an effect if the job status is `queued` or `running`.

The job status is set to `canceled` if the status was `running` beforehand and partial or preliminary results are available to be downloaded. Otherwise the status is set to `created`.

AUTHORIZATIONS: › *Bearer*

PATH PARAMETERS

job_id `required`
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

— **204** Processing the job has been successfully canceled.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  "id": "550e8400-e29b-11d4-a716-44665544000
  "code": "SampleError",
  "message": "Parameter 'sample' is missing.
- "links": [
    + { … }
  ]
}
```

Batch Jobs

Management of batch processing tasks (jobs) and their results.

List all batch jobs

GET `/jobs`

Lists all batch jobs submitted by a user.

It is **strongly RECOMMENDED** to keep the response size small by omitting all optional non-scalar values (i.e. arrays and objects) from objects in `jobs` (i.e. the `process` property). To get the full metadata for a job clients MUST request `GET /jobs/{job_id}`.

AUTHORIZATIONS: › *Bearer*

QUERY PARAMETERS

limit
integer `>= 1`
Example: `limit=10`
This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned.

Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources.

If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

⌄ **200** Array of job descriptions

RESPONSE SCHEMA: application/json

jobs › `required`
Array of objects (Batch Job)

links › `required`
Array of objects (links_pagination)
Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated.

If pagination is implemented, the following `rel` (relation) types apply:

1. `next` (REQUIRED): A link to the next page, except on the last page.
2. `prev` (OPTIONAL): A link to the previous page, except on the first page.
3. `first` (OPTIONAL): A link to the first page, except on the first page.
4. `last` (OPTIONAL): A link to the last page, except on the last page.

For additional relation types see also the lists of common relation types in openEO.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
- "jobs": [
    + { … }
  ],
- "links": [
    + { … }
  ]
}
```

Create a new batch job

POST `/jobs`

Creates a new batch processing task (job) from one or more (chained) processes at the back-end.

Processing the data does not start yet. The job status gets initialized as `created` by default.

AUTHORIZATIONS: › *Bearer*

REQUEST BODY SCHEMA: application/json
required

Specifies the job details, e.g. the user-defined process and billing details.

title
string or null (eo_title)
A short description to easily distinguish entities.

description
string or null <commonmark> (eo_description)
Detailed multi-line description to explain the entity.

CommonMark 0.29 syntax MAY be used for rich text representation.

process › `required`
object (Process Graph with metadata)
A process graph, optionally enriched with process metadata.

plan
string or null (billing_plan_null_default)
Default: `null`
The billing plan to process and charge the job or service with.

Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way:

- If a non-`null` value is given: Persist the `plan` that has been provided in the request.
- Otherwise:
  1. Persist the `default_plan` exposed through `GET /me`, if available.
  2. Persist the `default_plan` exposed through `GET /`, if available.
  3. If a single plan is exposed by the back-end, persist it.
  4. Otherwise, the back-end MUST throw a `BillingPlanMissing` error.

The resolved plan MUST be persisted permanently, regardless of any changes to the exposed billing plans in `GET /` in the future.

Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.

budget
number or null (budget) `>= 0`
Default: `null`
Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.

log_level
string (min_log_level_default)
Default: `"info"`
Enum: `"error"` `"warning"` `"info"` `"debug"`
The minimum severity level for log entries that the back-end stores for the processing request.

The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`.

The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.

property name* `additional property`
any
Additional back-end specific properties are allowed.

Responses

⌄ **201** The batch job has been created successfully.

RESPONSE HEADERS

Location `required`
string <uri>
Example: `"https://openeo.example/api/v1/jobs/123"`
Absolute URL to the newly created batch job.

The URL points to the metadata endpoint `GET /jobs/{job_id}` with the `{job_id}` being the unique identifier (ID) of the created batch job.

OpenEO-Identifier `required`
string (job_id) `^[\w\-\.~]+$`
Example: `"a3cca2b2aa1e3b5b"`
Per-back-end unique identifier of the batch job, generated by the back-end during creation. MUST match the specified pattern.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
  "title": "NDVI based on Sentinel-2",
  "description": "Deriving minimum NDVI meas
- "process": {
      "id": "ndvi",
      "summary": "string",
      "description": "string",
    + "parameters": [ … ],
    + "returns": { … },
    + "categories": [ … ],
      "deprecated": false,
      "experimental": false,
    + "exceptions": { … },
    + "examples": [ … ],
    + "links": [ … ],
    + "process_graph": { … }
  },
  "plan": "free",
  "budget": 100,
  "log_level": "warning",
  "property1": null,
  "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  "id": "550e8400-e29b-11d4-a716-44665544000
  "code": "SampleError",
  "message": "Parameter 'sample' is missing.
- "links": [
    + { … }
  ]
}
```

Modify a batch job

PATCH `/jobs/{job_id}`

Modifies an existing job at the back-end, but maintains the identifier. Changes can be grouped in a single request.

The job status does not change.

Jobs can only be modified when the job is not queued and not running. Otherwise, requests to this endpoint MUST be rejected with openEO error `JobLocked`.

AUTHORIZATIONS: › *Bearer*

PATH PARAMETERS

job_id `required`
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

REQUEST BODY SCHEMA: application/json
required

Specifies the job details to update.

title
string or null (eo_title)
A short description to easily distinguish entities.

description
string or null <commonmark> (eo_description)
Detailed multi-line description to explain the entity.

CommonMark 0.29 syntax MAY be used for rich text representation.

process ›
object (Process Graph with metadata)
A process graph, optionally enriched with process metadata.

plan
string or null (billing_plan_null)
The billing plan to process and charge the job or service with.

Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way if billing is supported:

- If a value is given and it is not `null`: Persist the `plan` that has been provided in the request.
- Otherwise, do not change the billing plan.

Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.

budget
number or null (budget_update) `>= 0`
Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null`.

log_level
string (min_log_level_update)
Enum: `"error"` `"warning"` `"info"` `"debug"`
Updates the minimum severity level for log entries that the back-end stores for the processing requests.

The back-end does not need to update existing log entries.

property name* `additional property`
any
Additional back-end specific properties are allowed.

Responses

— **204** Changes to the job applied successfully.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
  "title": "NDVI based on Sentinel-2",
  "description": "Deriving minimum NDVI meas
- "process": {
      "id": "ndvi",
      "summary": "string",
      "description": "string",
    + "parameters": [ … ],
    + "returns": { … },
    + "categories": [ … ],
      "deprecated": false,
      "experimental": false,
    + "exceptions": { … },
    + "examples": [ … ],
    + "links": [ … ],
    + "process_graph": { … }
  },
  "plan": "free",
  "budget": 100,
  "log_level": "warning",
  "property1": null,
  "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  "id": "550e8400-e29b-11d4-a716-44665544000
  "code": "SampleError",
  "message": "Parameter 'sample' is missing.
- "links": [
    + { … }
  ]
}
```

Full metadata for a batch job

GET `/jobs/{job_id}`

Lists all information about a submitted batch job.

AUTHORIZATIONS: › *Bearer*

PATH PARAMETERS

job_id `required`
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

⌄ **200** Full job information.

RESPONSE SCHEMA: application/json

id `required`
string (job_id) `^[\w\-\.~]+$`
Per-back-end unique identifier of the batch job, generated by the back-end during creation. MUST match the specified pattern.

title
string or null (eo_title)
A short description to easily distinguish entities.

description
string or null <commonmark> (eo_description)
Detailed multi-line description to explain the entity.

CommonMark 0.29 syntax MAY be used for rich text representation.

process › `required`
object (Process Graph with metadata)
A process graph, optionally enriched with process metadata.

status `required`
string
Default: `"created"`
Enum: `"created"` `"queued"` `"running"` `"canceled"` `"finished"` `"error"`
The current status of a batch job.

The following status changes can occur:

- `POST /jobs`: The status is initialized as `created`.
- `POST /jobs/{job_id}/results`: The status is set to `queued`, if processing does not start instantly.
  - Once the processing starts the status is set to `running`.
  - Once the data is available to download the status is set to `finished`.
  - Whenever an error occurs during processing, the status MUST be set to `error`.
- `DELETE /jobs/{job_id}/results`: The status is set to `canceled` if the status was `running` beforehand and partial or preliminary results are available to be downloaded. Otherwise the status is set to `created`.

progress
number `[ 0 .. 100 ]`
Indicates the process of a running batch job, in percent. Can also be set for a job which stopped due to an error or was canceled by the user. In this case, the value indicates the progress at which the job stopped. This property may not be available for the status codes `created` and `queued`. Submitted and queued jobs only allow the value `0`, finished jobs only allow the value `100`.

created `required`
string <date-time> (created)
Date and time of creation, formatted as a RFC 3339 date-time.

updated
string <date-time> (updated)
Date and time of the last status change, formatted as a RFC 3339 date-time.

plan
string (billing_plan)
The billing plan to process and charge the job or service with.

Billing plans MUST be handled in a *case insensitive* manner.

The plans can be retrieved from `GET /`, but the value returned here may not be in the list of plans any longer.

costs
number or null (money) `>= 0`
Default: `null`
An amount of money or credits. The value MUST be specified in the currency the back-end is working with. The currency can be retrieved by calling `GET /`. If no currency is set, this field MUST be `null`.

budget
number or null (budget) `>= 0`
Default: `null`
Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.

usage ›
object (Resource usage metrics)
Metrics about the resource usage of the batch job.

Back-ends are not expected to update the metrics while processing data, so the metrics can only be available after the job has finished or has stopped due to an error. For usage metrics during processing, metrics can better be added to the logs (e.g. `GET /jobs/{job_id}/logs`) with the same schema.

log_level
string (min_log_level_default)
Default: `"info"`
Enum: `"error"` `"warning"` `"info"` `"debug"`
The minimum severity level for log entries that the back-end stores for the processing request.

The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`.

The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.

links ›
Array of objects (Link)
Links related to this batch job such as links to invoices, log files or results.

Providing links with the following `rel` (relation) types is RECOMMENDED:

1. `monitor`: If logs are available, a link to the logs endpoint.
2. `result`: If batch job results are available, a link to the results endpoint.

The relation types `monitor` and `result` may occur for various batch job states:

1. `created`: When the batch job was executed before and has been reset to `created` after an update there could still be results and logs available until they get discarded by queueing the batch job again.
2. `finished`: The full log and results are expected to be available.
3. `error` / `canceled`: Partial results and logs may be available.

For more relation types see the lists of common relation types in openEO.

property name* `additional property`
any
You can list additional back-end specific properties here.

› **4XX**
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

› **5XX**
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  "property1": null,
  "property2": null,
  "id": "a3cca2b2aa1e3b5b",
  "title": "NDVI based on Sentinel-2",
  "description": "Deriving minimum NDVI meas
- "process": {
      "id": "ndvi",
      "summary": "string",
      "description": "string",
    + "parameters": [ … ],
    + "returns": { … },
    + "categories": [ … ],
      "deprecated": false,
      "experimental": false,
    + "exceptions": { … },
    + "examples": [ … ],
    + "links": [ … ],
    + "process_graph": { … }
  },
  "status": "running",
  "progress": 75.5,
  "created": "2017-01-01T09:32:12Z",
  "updated": "2017-01-01T09:36:18Z",
  "plan": "free",
  "costs": 12.98,
  "budget": 100,
- "usage": {
    + "cpu": { … },
    + "duration": { … },
    + "memory": { … },
    + "network": { … },
    + "storage": { … }
  },
  "log_level": "warning",
- "links": [
    + { … },
    + { … }
  ]
}
```

Delete a batch job

DELETE `/jobs/{job_id}`

Deletes all data related to a given batch job. Computations are stopped and computed results are deleted. This job will not generate additional costs for processing.

AUTHORIZATIONS: › *Bearer*

PATH PARAMETERS

job_id `required`
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

— **204** The job has been successfully deleted.

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  "id": "550e8400-e29b-11d4-a716-44665544000
  "code": "SampleError",
  "message": "Parameter 'sample' is missing.
- "links": [
    + { … }
  ]
}
```

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Get an estimate for a batch job

GET `/jobs/{job_id}/estimate`

Calculates an estimate for a batch job. Back-ends can decide to either calculate the duration, the costs, the size or a combination of them. Back-end providers MAY specify an expiry time for the estimate. Starting to process data afterwards MAY be charged at a higher cost. Costs do often not include download costs. Whether download costs are included or not can be indicated explicitly with the `downloads_included` flag. The estimate SHOULD be the upper limit of the costs, but back-end are free to use the field according to their terms of service. For some batch jobs it is not (easily) possible to estimate the costs reliably, e.g. if a UDF or ML model is part of the process. In this case, the server SHOULD return a `EstimateComplexity` error with HTTP status code 500.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

200
The estimated costs with regard to money, processing time and storage capacity. At least one of `costs`, `duration` or `size` MUST be provided.

RESPONSE SCHEMA: application/json

Any of: Batch Job Estimate | Batch Job Estimate | Batch Job Estimate

`costs` required
number or null (money) `>= 0`
Default: `null`
An amount of money or credits. The value MUST be specified in the currency the back-end is working with. The currency can be retrieved by calling `GET /`. If no currency is set, this field MUST be `null`.

`duration`
string
Estimated duration for the operation. Duration MUST be specified as an ISO 8601 duration.

`size`
integer
Estimated required storage capacity, i.e. the size of the generated files. Size MUST be specified in bytes.

`downloads_included`
integer or null
Default: `null`
Specifies how many full downloads of the processed data are included in the estimate. Set to `null` for unlimited downloads, which is also the default value.

`expires`
string <date-time>
Time until which the estimate is valid, formatted as a RFC 3339 date-time.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 | 4XX | 5XX

Content type
application/json

Copy

```
{
    "costs": 12.98,
    "duration": "P1Y2M10DT2H30M",
    "size": 157286400,
    "downloads_included": 5,
    "expires": "2020-11-01T00:00:00Z"
}
```

Logs for a batch job

GET `/jobs/{job_id}/logs`

Lists log entries for the batch job, usually for debugging purposes.

Back-ends can log any information that may be relevant for a user at any stage (status) of the batch job. Users can log information during data processing using respective processes such as `inspect`.

If requested consecutively, it is RECOMMENDED that clients use the offset parameter to get only the entries they have not received yet.

While pagination itself is OPTIONAL, the `offset` parameter is REQUIRED to be implemented by back-ends.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

QUERY PARAMETERS

`offset`
string
Example: `offset=log1234`
The last identifier (property `id` of a log entry) the client has received. If provided, the back-end MUST only send the entries that occurred after the specified identifier. If not provided or empty, the back-end MUST start with the first entry.

`level`
string
Default: `"debug"`
Enum: `"error"` `"warning"` `"info"` `"debug"`
Example: `level=error`
The minimum severity level for log entries that the back-end returns.

The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only return log entries with the level `warning` and `error`.

The default minimum log level is `debug`, which returns all log levels.

`limit`
integer `>= 1`
Example: `limit=10`
This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned.

Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources.

If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 Lists the requested log entries.

RESPONSE SCHEMA: application/json

`level`
string
Default: `"debug"`
Enum: `"error"` `"warning"` `"info"` `"debug"`
The minimum severity level for log entries that the back-end returns. This property MUST reflect the effective lowest `level` that may appear in the document, which is (if implemented) the highest level of:

1. the `log_level` specified by the user for the processing request.
2. the `level` specified by the user for the log request.

The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the logs will only contain entries with the level `warning` and `error`.

`logs` > required
Array of objects (Log Entry)
A chronological list of logs.

`links` > required
Array of objects (links_pagination)
Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated.

If pagination is implemented, the following `rel` (relation) types apply:

1. `next` (REQUIRED): A link to the next page, except on the last page.
2. `prev` (OPTIONAL): A link to the previous page, except on the first page.
3. `first` (OPTIONAL): A link to the first page, except on the first page.
4. `last` (OPTIONAL): A link to the last page, except on the last page.

For additional relation types see also the lists of common relation types in openEO.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 | 4XX | 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "level": "error",
  - "logs": [
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

List batch job results

GET `/jobs/{job_id}/results`

Lists signed URLs pointing to the processed files, usually after the batch job has finished. Back-ends may also point to intermediate results after the job has stopped due to an error or if the `partial` parameter has been set.

The response includes additional metadata. It is a valid STAC Item (if it has spatial and temporal references included) or a valid STAC Collection (supported since openEO API version 1.1.0). The assets to download are in both cases available in the property `assets` and have the same structure. Additional metadata is not strictly required to download the files, but are helpful for users to understand the data.

STAC Collections can either (1) add all assets as collection-level assets or (2) link to STAC Catalogs and STAC Items with signed URLs, which will provide a full STAC catalog structure a client has to go through. Option 2 is overall the better architectural choice and allows a fine-grained description of the processed data, but it is not compliant with previous versions of the openEO API. **To maintain backward compatibility, it is REQUIRED to still copy all assets in the STAC catalog structure into the collection-level assets.** This requirement is planned to be removed in openEO API version 2.0.0. A client can enforce that the server returns a GeoJSON through content negotiation with the media type `application/geo+json`, but the results may not contain very meaningful metadata aside from the assets.

Clients are RECOMMENDED to store this response and all potential sub-catalogs and items with the assets so that the downloaded data is then a self-contained STAC catalog user could publish easily with all the data and metadata.

URL signing is a way to protect files from unauthorized access with a key in the URL instead of HTTP header based authorization. The URL signing key is similar to a password and its inclusion in the URL allows to download files using simple GET requests supported by a wide range of programs, e.g. web browsers or download managers. Back-ends are responsible to generate the URL signing keys and to manage their appropriate expiration. The back-end MAY indicate an expiration time by setting the `expires` property in the reponse. Requesting this endpoint SHOULD always return non-expired URLs. Signed URLs that were generated for a previous request and already expired SHOULD NOT be reused, but regenerated with new expiration time. Signed URLs that expired MAY return the openEO error `ResultLinkExpired`.

Adding a link with relation type `canonical` to the STAC Item or STAC Collection is STRONGLY RECOMMENDED (see the `links` property for details).

If processing has not finished yet and the `partial` parameter is not set to `true` requests to this endpoint MUST be rejected with openEO error `JobNotFinished`.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

QUERY PARAMETERS

`partial`
boolean
Default: `false`
If set to `true`, the results endpoint returns incomplete results while still running. Enabling this parameter requires to indicate the status of the batch job in the STAC metadata by setting the `openeo:status`.

Responses

200
Valid download links have been returned. The download links does not necessarily need to be located under the API base url.

RESPONSE HEADERS

`OpenEO-Costs`
number or null (money) `>= 0`
Default: `null`
Example: `"12.98"`
Specifies the costs for fully downloading the data **once**, i.e. this header MAY change in subsequent calls. It MUST be set to `0` if the requester is the owner of the job and still has free downloads included in his processing charges estimated by `GET /jobs/{job_id}/estimate`. If a requester other than the owner is requesting the data of a shared job this header indicates the costs for the requester.

RESPONSE SCHEMA: application/json

One of: Batch Job Results Response as STAC Item | Batch Job Results Response as STAC Collection

`stac_version` required
string (stac_version) `^(0\.9.\d+|1\.\d+.\d+)`
The version of the STAC specification, which MAY not be equal to the STAC API version. Supports versions 0.9.x and 1.x.x.

`stac_extensions` >
Array of Reference to a JSON Schema (string) or Reference to a core extension (STAC < 1.0.0-rc.1 only) (string) (stac_extensions) `unique`
A list of implemented STAC extensions. The list contains URLs to the JSON Schema files it can be validated against. For STAC < 1.0.0-rc.1 shortcuts such as `sar` can be used instead of the schema URL.

`id` required
string (job_id) `^[\w\-\.~]+$`
Per-back-end unique identifier of the batch job, generated by the back-end during creation. MUST match the specified pattern.

`type` required
string
Value: `"Feature"`
The GeoJSON type that applies to this metadata document, which MUST always be a "Feature" according to the STAC specification. This type does **not** describe the spatial data type of the assets.

`bbox` >
Array of 4 elements (numbers) or Array of 6 elements (numbers) (bbox)
Each bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth):

- West (lower left corner, coordinate axis 1)
- South (lower left corner, coordinate axis 2)
- Base (optional, minimum value, coordinate axis 3)
- East (upper right corner, coordinate axis 1)
- North (upper right corner, coordinate axis 2)
- Height (optional, maximum value, coordinate axis 3)

The coordinate reference system of the values is WGS 84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84).

For WGS 84 longitude/latitude the values are in most cases the sequence of minimum longitude, minimum latitude, maximum longitude and maximum latitude.

However, in cases where the box spans the antimeridian the first value (west-most box edge) is larger than the third value (east-most box edge).

If the vertical axis is included, the third and the sixth number are the bottom and the top of the 3-dimensional bounding box.

`geometry` > required
object or null (GeoJSON Geometry)
Defines the full footprint of the asset represented by this item as GeoJSON Geometry.

Results without a known location can set this value to `null`.

`properties` > required
object (Item Properties)
MAY contain additional properties other than the required property `datetime`, e.g. custom properties or properties from the STAC specification or STAC extensions.

`assets` > required
object (Assets)
Dictionary of asset objects for data that can be downloaded, each with a unique key. The keys MAY be used by clients as file names.

`links` > required
Array of objects (Link)
Links related to this batch job result, e.g. a link to an invoice, additional log files or external documentation.

The links MUST NOT contain links to the processed and downloadable data. Instead specify these in the `assets` property. Clients MUST NOT download the data referenced in the links by default.

It is **strongly recommended** to add a link with relation type `canonical`, which points to this STAC document using a signed URL. This way the STAC metadata can be used by non-openEO clients without additional authentication steps.

For relation types see the lists of common relation types in openEO.

> 424
The request can not be fulfilled as the batch job failed. This request will deliver the last error message that was produced by the batch job.

This HTTP code MUST be sent only when the job `status` is `error`.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 | 424 | 4XX | 5XX

Content type
application/json

Example
Batch Job Results Response as STAC Item

Copy Expand all Collapse all

```
{
    "stac_version": "1.0.0",
  - "stac_extensions": [
        "https://openeo.example/stac/custom-ext
    ],
    "id": "a3cca2b2aa1e3b5b",
    "type": "Feature",
  - "bbox": [
        -180,
        -90,
        180,
        90
    ],
  - "geometry": {
        "type": "Polygon",
      + "coordinates": [ … ]
    },
  - "properties": {
        "datetime": "2019-08-24T14:15:22Z",
        "start_datetime": "2019-08-24T14:15:22
        "end_datetime": "2019-08-24T14:15:22Z",
        "title": "NDVI based on Sentinel-2",
        "description": "Deriving minimum NDVI m
        "license": "Apache-2.0",
      + "providers": [ … ],
        "created": "2017-01-01T09:32:12Z",
        "updated": "2017-01-01T09:36:18Z",
        "expires": "2020-11-01T00:00:00Z",
        "openeo:status": "running"
    },
  - "assets": {
      + "preview.png": { … },
      + "process.json": { … },
      + "1.tif": { … },
      + "2.tif": { … },
      + "inspire.xml": { … }
    },
  - "links": [
      + { … }
    ]
}
```

Start processing a batch job

POST `/jobs/{job_id}/results`

Adds a batch job to the processing queue to compute the results.

The result will be stored in the format specified in the process. To specify the format use a process such as `save_result`.

The job status is set to `queued`, if processing does not start instantly. The same applies if the job status is `canceled`, `finished`, or `error`, which restarts the job and discards previous results if the back-end does not reject the request with an error. Clients SHOULD warn users and ask for confirmation if results may get discarded.

- Once the processing starts the status is set to `running`.
- Once the data is available to download the status is set to `finished`.
- Whenever an error occurs during processing, the status MUST be set to `error`.

This endpoint has no effect if the job status is already `queued` or `running`. In particular, it does not restart a running job. To restart a queued or running job, processing MUST have been canceled.

Back-ends SHOULD reject queueing jobs with openEO error `PaymentRequired`, if the back-end is able to detect that the budget is too low to fully process the request. Alternatively, back-ends MAY provide partial results once reaching the budget. If none of the alternatives is feasible, the results are discarded. Thus, client SHOULD warn users that reaching the budget may lead to partial or no results at all.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

— **202** The creation of the resource has been queued successfully.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX | 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Cancel processing a batch job

DELETE `/jobs/{job_id}/results`

Cancels all related computations for this job at the back-end. It will stop generating additional costs for processing.

A subset of processed results may be available for downloading depending on the state of the job at the time it was canceled.

Results MUST NOT be deleted until the job processing is started again or the job is completely deleted through a request to `DELETE /jobs/{job_id}`.

This endpoint only has an effect if the job status is `queued` or `running`.

The job status is set to `canceled` if the status was `running` beforehand and partial or preliminary results are available to be downloaded. Otherwise the status is set to `created`.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`job_id` required
string (job_id) `^[\w\-\.~]+$`
Example: `a3cca2b2aa1e3b5b`
Identifier of the batch job.

Responses

— **204** Processing the job has been successfully canceled.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX | 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Secondary Services

On-demand access to data using other web service protocols.

Supported secondary web service protocols

GET `/service_types`

Lists supported secondary web service protocols such as OGC WMS, OGC WCS, OGC API - Features or XYZ tiles. The response is an object of all available secondary web service protocols with their supported configuration settings and expected process parameters.

- The configuration settings for the service SHOULD be defined upon creation of a service and the service will be set up accordingly.
- The process parameters SHOULD be referenced (with a `from_parameter` reference) in the user-defined process that is used to compute web service results. The appropriate arguments MUST be provided to the user-defined process, usually at runtime from the context of the web service. For example, a map service such as a WMS would need to inject the spatial extent into the user-defined process so that the back-end can compute the corresponding tile correctly.

To improve interoperability between back-ends common names for the services SHOULD be used, e.g. the abbreviations used in the official OGC Schema Repository for the respective services.

Service names MUST be accepted in a *case insensitive* manner throughout the API.

AUTHORIZATIONS: None or *Bearer*

Responses

200
An object with a map containing all service names as keys and an object that defines supported configuration settings and process parameters.

RESPONSE SCHEMA: application/json

`Service Name*` > additional property
object (Service Type)

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request usually does not respond with HTTP status codes 401 and 403 due to missing authorization. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 | 4XX | 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "WMS": {
        "title": "OGC Web Map Service",
      + "configuration": { … },
      + "process_parameters": [ … ],
      + "links": [ … ]
    },
  - "OGCAPI-FEATURES": {
        "title": "OGC API - Features",
        "description": "Exposes a OGC API - Fea
      + "configuration": { … },
        "process_parameters": [ ],
      + "links": [ … ]
    }
}
```

List all web services

GET `/services`

Lists all secondary web services submitted by a user.

It is **strongly RECOMMENDED** to keep the response size small by omitting all optional non-scalar values (i.e. arrays and objects) from objects in `services` (i.e. the `process`, `configuration` and `attributes` properties). To get the full metadata for a secondary web service clients MUST request `GET /services/{service_id}`.

AUTHORIZATIONS: *Bearer*

QUERY PARAMETERS

`limit`
integer `>= 1`
Example: `limit=10`
This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned.

Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources.

If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

200 Array of secondary web service descriptions

RESPONSE SCHEMA: application/json

`services` > required
Array of objects (Secondary Web Service)

`links` > required
Array of objects (links_pagination)
Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated.

If pagination is implemented, the following `rel` (relation) types apply:

1. `next` (REQUIRED): A link to the next page, except on the last page.
2. `prev` (OPTIONAL): A link to the previous page, except on the first page.
3. `first` (OPTIONAL): A link to the first page, except on the first page.
4. `last` (OPTIONAL): A link to the last page, except on the last page.

For additional relation types see also the lists of common relation types in openEO.

> 4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

Response samples

200 | 4XX | 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "services": [
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Publish a new service

POST `/services`

Creates a new secondary web service such as a OGC WMS, OGC WCS, OGC API - Features or XYZ tiles.

The secondary web service SHOULD process the underlying data on demand, based on process parameters provided to the user-defined process (through `from_parameter` references) at run-time, for example for the spatial/temporal extent, resolution, etc. The available process parameters are specified per service type at `GET /service_types`.

Note: Costs incurred by shared secondary web services are usually paid by the owner, but this depends on the service type and whether it supports charging fees or not.

AUTHORIZATIONS: *Bearer*

REQUEST BODY SCHEMA: application/json
required

The base data required to create the secondary web service.

`title`	string or null (eo_title) A short description to easily distinguish entities.
`description`	string or null <commonmark> (eo_description) Detailed multi-line description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation.
`process` > required	object (Process Graph with metadata) A process graph, optionally enriched with process metadata.
`type` required	string (service_type) Definition of the service type to access result data. All available service types can be retrieved via `GET /service_types`. Service types MUST be accepted in a *case insensitive* manner.
`enabled`	boolean Default: `true` Describes whether a secondary web service is responding to requests (true) or not (false). Disabled services do not produce any costs.
`configuration`	object (Service Configuration) Map of configuration settings, i.e. the setting names supported by the secondary web service combined with actual values. See `GET /service_types` for supported configuration settings. For example, this could specify the required version of the service, visualization details or any other service dependent configuration.
`plan`	string or null (billing_plan_null_default) Default: `null` The billing plan to process and charge the job or service with. Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way: • If a non-`null` value is given: Persist the `plan` that has been provided in the request. • Otherwise: 1. Persist the `default_plan` exposed through `GET /me`, if available. 2. Persist the `default_plan` exposed through `GET /`, if available. 3. If a single plan is exposed by the back-end, persist it. 4. Otherwise, the back-end MUST throw a `BillingPlanMissing` error. The resolved plan MUST be persisted permanently, regardless of any changes to the exposed billing plans in `GET /` in the future. Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.
`budget`	number or null (budget) >= 0 Default: `null` Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.
`log_level`	string (min_log_level_default) Default: `"info"` Enum: `"error"` `"warning"` `"info"` `"debug"` The minimum severity level for log entries that the back-end stores for the processing request. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`. The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.
`property name*` additional property	any Additional back-end specific properties are allowed.

Responses

˅ **201** The service has been created successfully.

RESPONSE HEADERS

`Location` required	string <uri> Example: `"https://openeo.example/api/v1/services/123"` Absolute URL to the newly created service. The URL points to the metadata endpoint `GET /services/{service_id}` with the `{service_id}` being the unique identifier (ID) of the created service. MUST NOT point to the actual instance (e.g. WMTS base URL) of the service. The URL to the instance is made available by the metadata endpoint in the property `url`.
`OpenEO-Identifier` required	string (service_id) `^[\w\-\.~]+$` Example: `"wms-a3cca9"` A per-back-end unique identifier of the secondary web service, generated by the back-end during creation. MUST match the specified pattern.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
    "title": "NDVI based on Sentinel-2",
    "description": "Deriving minimum NDVI meas
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
    "type": "wms",
    "enabled": true,
  - "configuration": {
        "version": "1.3.0"
    },
    "plan": "free",
    "budget": 100,
    "log_level": "warning",
    "property1": null,
    "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Modify a service

PATCH `/services/{service_id}`

Modifies an existing secondary web service at the back-end, but maintains the identifier. Changes can be grouped into a single request.

User MUST create a new service to change the service type.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`service_id` required	string (service_id) `^[\w\-\.~]+$` Example: `wms-a3cca9` Identifier of the secondary web service.

REQUEST BODY SCHEMA: application/json
required

The data to change for the specified secondary web service.

`title`	string or null (eo_title) A short description to easily distinguish entities.
`description`	string or null <commonmark> (eo_description) Detailed multi-line description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation.
`process` >	object (Process Graph with metadata) A process graph, optionally enriched with process metadata.
`enabled`	boolean (service_enabled) Describes whether a secondary web service is responding to requests (true) or not (false). Disabled services do not produce any costs.
`configuration`	object (Service Configuration) Map of configuration settings, i.e. the setting names supported by the secondary web service combined with actual values. See `GET /service_types` for supported configuration settings. For example, this could specify the required version of the service, visualization details or any other service dependent configuration.
`plan`	string or null (billing_plan_null) The billing plan to process and charge the job or service with. Billing plans MUST be accepted in a *case insensitive* manner. Back-ends MUST resolve the billing plan in the following way if billing is supported: • If a value is given and it is not `null`: Persist the `plan` that has been provided in the request. • Otherwise, do not change the billing plan. Billing plans not on the list of available plans MUST be rejected with openEO error `BillingPlanInvalid`.
`budget`	number or null (budget_update) >= 0 Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null`.
`log_level`	string (min_log_level_update) Enum: `"error"` `"warning"` `"info"` `"debug"` Updates the minimum severity level for log entries that the back-end stores for the processing requests. The back-end does not need to update existing log entries.
`property name*` additional property	any Additional back-end specific properties are allowed.

Responses

— **204** Changes to the service were applied successfully.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
    "title": "NDVI based on Sentinel-2",
    "description": "Deriving minimum NDVI meas
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
    "enabled": true,
  - "configuration": {
        "version": "1.3.0"
    },
    "plan": "free",
    "budget": 100,
    "log_level": "warning",
    "property1": null,
    "property2": null
}
```

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Full metadata for a service

GET `/services/{service_id}`

Lists all information about a secondary web service.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`service_id` required	string (service_id) `^[\w\-\.~]+$` Example: `wms-a3cca9` Identifier of the secondary web service.

Responses

˅ **200** Details of the created service

RESPONSE SCHEMA: application/json

`id` required	string (service_id) `^[\w\-\.~]+$` A per-back-end unique identifier of the secondary web service, generated by the back-end during creation. MUST match the specified pattern.
`title`	string or null (eo_title) A short description to easily distinguish entities.
`description`	string or null <commonmark> (eo_description) Detailed multi-line description to explain the entity. CommonMark 0.29 syntax MAY be used for rich text representation.
`url` required	string <uri> URL at which the secondary web service is accessible. Does not necessarily need to be located within the API.
`type` required	string (service_type) Definition of the service type to access result data. All available service types can be retrieved via `GET /service_types`. Service types MUST be accepted in a *case insensitive* manner.
`enabled` required	boolean (service_enabled) Describes whether a secondary web service is responding to requests (true) or not (false). Disabled services do not produce any costs.
`process` > required	object (Process Graph with metadata) A process graph, optionally enriched with process metadata.
`configuration` required	object (Service Configuration) Map of configuration settings, i.e. the setting names supported by the secondary web service combined with actual values. See `GET /service_types` for supported configuration settings. For example, this could specify the required version of the service, visualization details or any other service dependent configuration.
`attributes` required	object (Secondary Web Service Attributes) Additional attributes of the secondary web service, e.g. available layers for a WMS instance based on the bands in the underlying GeoTiff.
`created`	string <date-time> (created) Date and time of creation, formatted as a RFC 3339 date-time.
`plan`	string (billing_plan) The billing plan to process and charge the job or service with. Billing plans MUST be handled in a *case insensitive* manner. The plans can be retrieved from `GET /`, but the value returned here may not be in the list of plans any longer.
`costs`	number or null (money) >= 0 Default: `null` An amount of money or credits. The value MUST be specified in the currency the back-end is working with. The currency can be retrieved by calling `GET /`. If no currency is set, this field MUST be `null`.
`budget`	number or null (budget) >= 0 Default: `null` Maximum amount of costs the request is allowed to produce. The value MUST be specified in the currency of the back-end. No limits apply, if the value is `null` or the back-end has no currency set in `GET /`.
`usage` >	object (Resource usage metrics) Metrics about the resource usage of the secondary web service. Back-ends are not expected to update the metrics in real-time. For detailed usage metrics for individual processing steps, metrics can be added to the logs (e.g. `GET /jobs/{job_id}/logs`) with the same schema.
`log_level`	string (min_log_level_default) Default: `"info"` Enum: `"error"` `"warning"` `"info"` `"debug"` The minimum severity level for log entries that the back-end stores for the processing request. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only store log entries with the level `warning` and `error`. The default minimum log level is `info`. Users need to specifically set this property to `debug` to capture *all* log entries. It is RECOMMENDED that users set the level at least to "warning" in production workflows.
`property name*` additional property	any You can list additional back-end specific properties here.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "property1": null,
    "property2": null,
    "id": "wms-a3cca9",
    "title": "NDVI based on Sentinel-2",
    "description": "Deriving minimum NDVI meas
    "url": "https://openeo.example/wms/wms-a3c
    "type": "wms",
    "enabled": true,
  - "process": {
        "id": "ndvi",
        "summary": "string",
        "description": "string",
      + "parameters": [ … ],
      + "returns": { … },
      + "categories": [ … ],
        "deprecated": false,
        "experimental": false,
      + "exceptions": { … },
      + "examples": [ … ],
      + "links": [ … ],
      + "process_graph": { … }
    },
  - "configuration": {
        "version": "1.3.0"
    },
  - "attributes": {
      + "layers": [ … ]
    },
    "created": "2017-01-01T09:32:12Z",
    "plan": "free",
    "costs": 12.98,
    "budget": 100,
  - "usage": {
      + "cpu": { … },
      + "duration": { … },
      + "memory": { … },
      + "network": { … },
      + "storage": { … }
    },
    "log_level": "warning"
}
```

Delete a service

DELETE `/services/{service_id}`

Deletes all data related to this secondary web service. Computations are stopped, computed results are deleted, and access to this service is no longer possible. This service will not generate additional costs.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`service_id` required	string (service_id) `^[\w\-\.~]+$` Example: `wms-a3cca9` Identifier of the secondary web service.

Responses

— **204** The service has been successfully deleted.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

Logs for a secondary service

GET `/services/{service_id}/logs`

Lists log entries for the secondary service, usually for debugging purposes. Back-ends can log any information that may be relevant for a user. Users can log information during data processing using respective processes such as `inspect`. If requested consecutively while the secondary service is enabled, it is RECOMMENDED that clients use the offset parameter to get only the entries they have not received yet. While pagination itself is OPTIONAL, the `offset` parameter is REQUIRED to be implemented by back-ends.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`service_id` required	string (service_id) `^[\w\-\.~]+$` Example: `wms-a3cca9` Identifier of the secondary web service.

QUERY PARAMETERS

`offset`	string Example: `offset=log1234` The last identifier (property `id` of a log entry) the client has received. If provided, the back-end MUST only send the entries that occurred after the specified identifier. If not provided or empty, the back-end MUST start with the first entry.
`level`	string Default: `"debug"` Enum: `"error"` `"warning"` `"info"` `"debug"` Example: `level=error` The minimum severity level for log entries that the back-end returns. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the back-end will only return log entries with the level `warning` and `error`. The default minimum log level is `debug`, which returns all log levels.
`limit`	integer >= 1 Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

˅ **200** Lists the requested log entries.

RESPONSE SCHEMA: application/json

`level`	string Default: `"debug"` Enum: `"error"` `"warning"` `"info"` `"debug"` The minimum severity level for log entries that the back-end returns. This property MUST reflect the effective lowest `level` that may appear in the document, which is (if implemented) the highest level of: 1. the `log_level` specified by the user for the processing request. 2. the `level` specified by the user for the log request. The order of the levels is as follows (from low to high severity): `debug`, `info`, `warning`, `error`. That means if `warning` is set, the logs will only contain entries with the level `warning` and `error`.
`logs` > required	Array of objects (Log Entry) A chronological list of logs.
`links` > required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "level": "error",
  - "logs": [
      + { … }
    ],
  - "links": [
      + { … }
    ]
}
```

File Storage

Management of user-uploaded assets and processed data.

List all files in the workspace

GET `/files`

Lists all user-uploaded files that are stored at the back-end.

AUTHORIZATIONS: *Bearer*

QUERY PARAMETERS

`limit`	integer >= 1 Example: `limit=10` This parameter enables pagination for the endpoint and specifies the maximum number of elements that arrays in the top-level object (e.g. collections, processes, batch jobs, secondary services, log entries, etc.) are allowed to contain. The `links` array MUST NOT be paginated like the resources, but instead contain links related to the paginated resources or the pagination itself (e.g. a link to the next page). If the parameter is not provided or empty, all elements are returned. Pagination is OPTIONAL: back-ends or clients may not support it. Therefore, it MUST be implemented in a way that clients not supporting pagination get all resources regardless. Back-ends not supporting pagination MUST return all resources. If the response is paginated, the `links` array MUST be used to communicate the links for browsing the pagination with predefined `rel` types. See the `links` array schema for supported `rel` types. Back-end implementations can, unless specified otherwise, use any kind of pagination technique, depending on what is supported best by their infrastructure: page-based, offset-based, token-based or something else. The clients SHOULD use whatever is specified in the links with the corresponding `rel` types.

Responses

˅ **200**
Flattened file tree with path relative to the user's root directory and some basic properties such as the file size and the timestamp of the last modification. All properties except the name are optional. Folders MUST NOT be listed separately so each element in the list MUST be a downloadable file.

RESPONSE SCHEMA: application/json

`files` > required	Array of objects (Workspace File)
`links` > required	Array of objects (links_pagination) Links related to this list of resources, for example links for pagination or alternative formats such as a human-readable HTML version. The links array MUST NOT be paginated. If pagination is implemented, the following `rel` (relation) types apply: 1. `next` (REQUIRED): A link to the next page, except on the last page. 2. `prev` (OPTIONAL): A link to the previous page, except on the first page. 3. `first` (OPTIONAL): A link to the first page, except on the first page. 4. `last` (OPTIONAL): A link to the last page, except on the last page. For additional relation types see also the lists of common relation types in openEO.

> 4XX

The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

> 5XX

The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
  - "files": [
      + { … },
      + { … },
      + { … }
    ],
    "links": [ ]
}
```

Download a file from the workspace

GET `/files/{path}`

Offers a file from the user workspace for download. The file is identified by its path relative to the user's root directory. If a folder is specified as path a `FileOperationUnsupported` error MUST be sent as response.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```

`path` required

string

Examples:

- `borders/europe.geojson` - A path without special chars. It describes a file `europe.geojson` in a folder called `borders`.
- `europe/%C3%B6sterreich/m%C3%BCnster.shp` - A path with special chars. It describes a file `münster.shp` in folders called `europe` and `österreich`.

Path of the file, relative to the user's root directory. MAY include folders, but MUST not include relative references such as `.` and `..`.

Folder and file names in the path MUST be url-encoded. The path separator `/` and the file extension separator `.` MUST NOT be url-encoded.

The URL-encoding may be shown incorrectly in rendered versions due to OpenAPI 3 not supporting path parameters which contain slashes. This may also lead to OpenAPI validators not validating paths containing folders correctly.

Responses

200 A file from the workspace.

RESPONSE SCHEMA: application/octet-stream

string <binary>

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Upload a file to the workspace

PUT `/files/{path}`

Uploads a new file to the given path or updates an existing file if a file at the path exists.

Folders are created once required by a file upload. Empty folders can not be created.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`path` required

string

Examples:

- `borders/europe.geojson` - A path without special chars. It describes a file `europe.geojson` in a folder called `borders`.
- `europe/%C3%B6sterreich/m%C3%BCnster.shp` - A path with special chars. It describes a file `münster.shp` in folders called `europe` and `österreich`.

Path of the file, relative to the user's root directory. MAY include folders, but MUST not include relative references such as `.` and `..`.

Folder and file names in the path MUST be url-encoded. The path separator `/` and the file extension separator `.` MUST NOT be url-encoded.

The URL-encoding may be shown incorrectly in rendered versions due to OpenAPI 3 not supporting path parameters which contain slashes. This may also lead to OpenAPI validators not validating paths containing folders correctly.

REQUEST BODY SCHEMA: application/octet-stream
required

string <binary>

Responses

200 The file has been uploaded successfully.

RESPONSE SCHEMA: application/json

`path` required

string `^[^/ \:'"][^ \:'"]*$`

Path of the file, relative to the root directory of the user's server-side workspace. MUST NOT start with a slash `/` and MUST NOT be url-encoded.

The Windows-style path name component separator `\` is not supported, always use `/` instead.

Note: The pattern only specifies a minimal subset of invalid characters. The back-ends MAY enforce additional restrictions depending on their OS/environment.

`size`

integer

File size in bytes.

`modified`

string <date-time>

Date and time the file has lastly been modified, formatted as a RFC 3339 date-time.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

200 4XX 5XX

Content type
application/json

Copy

```
{
    "path": "folder/file.txt",
    "size": 1024,
    "modified": "2018-01-03T10:55:29Z"
}
```

Delete a file from the workspace

DELETE `/files/{path}`

Deletes an existing user-uploaded file specified by its path. Resulting empty folders MUST be deleted automatically.

Back-ends MAY support deleting folders including its files and sub-folders. If not supported by the back-end a `FileOperationUnsupported` error MUST be sent as response.

AUTHORIZATIONS: *Bearer*

PATH PARAMETERS

`path` required

string

Examples:

- `borders/europe.geojson` - A path without special chars. It describes a file `europe.geojson` in a folder called `borders`.
- `europe/%C3%B6sterreich/m%C3%BCnster.shp` - A path with special chars. It describes a file `münster.shp` in folders called `europe` and `österreich`.

Path of the file, relative to the user's root directory. MAY include folders, but MUST not include relative references such as `.` and `..`.

Folder and file names in the path MUST be url-encoded. The path separator `/` and the file extension separator `.` MUST NOT be url-encoded.

The URL-encoding may be shown incorrectly in rendered versions due to OpenAPI 3 not supporting path parameters which contain slashes. This may also lead to OpenAPI validators not validating paths containing folders correctly.

Responses

— **204** The file has been successfully deleted at the back-end.

4XX
The request can not be fulfilled due to an error on client-side, i.e. the request is invalid. The client SHOULD NOT repeat the request without modifications.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. This request MUST respond with HTTP status codes 401 if authorization is required or 403 if the authorization failed or access is forbidden in general to the authenticated user. HTTP status code 404 SHOULD be used if the value of a path parameter is invalid.

See also:

- Error Handling in the API in general.
- Common Error Codes

5XX
The request can not be fulfilled due to an error at the back-end. The error is never the client's fault and therefore it is reasonable for the client to retry the exact same request that triggered this response.

The response body SHOULD contain a JSON error object. MUST be any HTTP status code specified in RFC 7231. See also:

- Error Handling in the API in general.
- Common Error Codes

Response samples

4XX 5XX

Content type
application/json

Copy Expand all Collapse all

```
{
    "id": "550e8400-e29b-11d4-a716-44665544000
    "code": "SampleError",
    "message": "Parameter 'sample' is missing.
  - "links": [
      + { … }
    ]
}
```