Package: openeo (via r-universe)

September 13, 2024

Type Package

Title Client Interface for 'openEO' Servers

Version 1.3.1

Description Access data and processing functionalities of 'openEO' compliant back-ends in R.

Depends R (>= 3.5.0)

Suggests tibble, testthat, knitr, stars, pkgdown, rmarkdown, kableExtra, DT, terra, magrittr

Imports jsonlite, httr2 (>= 0.2.2), methods, R6, lubridate, base64enc, sf, IRdisplay, htmltools, rlang

Encoding UTF-8

LazyData false

BugReports https://github.com/Open-EO/openeo-r-client/issues

URL https://open-eo.github.io/openeo-r-client/, https://github.com/Open-EO/openeo-r-client

RoxygenNote 7.3.1

VignetteBuilder knitr

License Apache License (>= 2)

Collate 'argument_types.R' 'authentication.R' 'zzz.R' 'capabilities-mapping.R' 'process_graph_building.R' 'utilities.R' 'client.R' 'debugging.R' 'jobs.R' 'services.R' 'user_defined_processes.R' 'coerce-functions.R' 'collection-functions.R' 'collections.R' 'ops.R' 'predefined_processes.R' 'print-functions.R' 'sample_data.R' 'server_metadata.R' 'spatial.R' 'udf.R' 'user.R' 'viewer.R'

Roxygen list(markdown = TRUE)

Repository https://open-eo.r-universe.dev

RemoteUrl https://github.com/open-eo/openeo-r-client

RemoteRef HEAD

RemoteSha e2612a1d8429725e681d92a2ae5167a5b44585a1

Contents

`active_connection` *Active Connection*

Description

The function gets or sets the currently active connection to an openEO service. Usually, the active connection is set when calling the `connect()` function. Just the last connection is set as active. An application for the active connection is the optional connection within all the functions that interact with the openEO service and require a connection. If the connection is omitted in the function, this function is called in order to try to fetch a connection. If you want to operate on multiple services at once, you should use an explicit connection.

Usage

```
active_connection(con = NULL)
```

Arguments

`con`	optional `OpenEOClient()` to set, if omitted or NULL the currently active connection is returned

Value

OpenEOClient()

See Also

connect()

Examples

```
## Not run:
# Note: all URLs and credentials are arbitrary
con1 = connect("https://first.openeo-backend.com")
con2 = connect("https://second.openeo-backend.com")

active_connection() # this will be con2, the last connected backend

active_connection(con = con1) # sets the first connection as active, so it does not have to
# be passed to all functions

active_connection() # this will now return the previous set connection con1

## End(Not run)
```

AnyOf *AnyOf*

Description

Inheriting from Argument() in order to represent an argument choice object. Multiple types can be stated, but at least one data type has to be picked. In a JSON-schema this is often used to make objects nullable - meaning that they allow NULL as value. The AnyOf parameter is resolved into a simple nullable argument if this applies.

Value

Object of R6Class() representing an argument choice object.

Methods

$getChoice() returns a list of Argument() that are allowed

$isNullable returns TRUE if only one element is in the choice that is not "null"

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

---

api_versions *Returns the supported openEO API versions*

Description

The function queries the back-end for its supported versions. The endpoint /.well-known/openeo is called on the given host URL and the JSON result is coerced into a tibble.

Usage

```
api_versions(url)
```

Arguments

url	the URL as string pointing to the base host of the back-end

Value

a data.frame or a tibble containing all supported API versions of the back-end

---

Argument *Argument class*

Description

This class inherits all fields and functions from Parameter() adds the functionality to manage a value. This includes getter/setter, validation and serialization. Since this is the parent class for the type specific argument classes, the inheriting classes implement their own version of the private functions $typeCheck() and $typeSerialization().

Value

Object of R6Class() representing an argument.

Methods

`$setValue(value)` Assigns a value to this argument

`$getValue()` Returns the value of this argument

`$serialize()` returns a list representation of a openEO argument

`$validate()` return TRUE if the parameter is validated positively by the type check

`$isEmpty()` returns TRUE if the value is set

`$getProcess()` returns the process this parameter belongs to

`$setProcess(p)` sets the owning process for this parameter

Arguments

`value` The value for this argument.

`p` An object of class 'Process' or inheriting like 'ProcessNode'

`Array` *Array*

Description

Inheriting from `Argument()` in order to represent an array of a single data type.

Value

Object of `R6Class()` representing a single valued array.

Methods

`$getMinItems` returns the minimum number of items

`$getMaxItems` returns the maximum number of items

`$setMinItems(value)` sets the minimum number of items

`$setMaxItems(value)` sets the maximum number of items

`$getItemSchema` returns the item schema of the items in the array

`$setItemSchema(value)` sets the schema for the items in the array

Arguments

`value` either a number describing the minimum and maximum number of elements in an array or the parsed JSON schema of a single item in the array

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

as.bbox *coerce to bbox*

Description

A coercion function for extracting a 'bbox' object that can usualy be obtained by `sf::st_bbox()`. This coercion function was created to easily extract the boúnding box from the openeos argument objects `BoundingBox()` and `GeoJson()`.

Usage

```
`as.bbox.bounding-box`(from)

as.bbox.geojson(from)
```

Arguments

`from`	a `BoundingBox()` argument object or a `GeoJson()` argument object

Value

a bbox object from `sf::st_bbox()`

as.data.frame *Coercions into data.frame objects*

Description

The openEO package offers functions to transform list objects obtained from JSON into data.frames. This is mostly applied in list_* functions.

Usage

```
## S3 method for class 'JobList'
as.data.frame(x, ...)

## S3 method for class 'ServiceList'
as.data.frame(x, ...)

## S3 method for class 'BandList'
as.data.frame(x, ...)

## S3 method for class 'CollectionList'
as.data.frame(x, ...)

## S3 method for class 'VersionsList'
```

```
as.data.frame(x, ...)

## S3 method for class 'FileFormatList'
as.data.frame(x, ...)
```

Arguments

x	the list object that will be coerced
...	potentially additional parameters to pass on to internal functions like 'extract'

Details

The parameter 'extract' is used as an additional parameter to extract specific values of the output list / json. The value for the parameters is a vector of character like c('id','title')

Value

a data.frame

---

as.Graph *Coercion into Graph*

Description

Creates a Graph object from a ProcessNode(), function or ProcessInfo (Exchange object for predefined and stored user-defined processes).

Usage

```
as.Graph.ProcessNode(from)

as.Graph.function(from)

as.Graph.ProcessInfo(from)

as.Graph.Process(from)
```

Arguments

from	the source from which to coerce (ProcessNode, function or ProcessInfo)

Details

Those pure Graph objects shall only be used internally. If you want to use this information to directly interact with the back-end via JSON please use as.Process(). This function might be removed from the package function export in the future.

Value

Graph()

---

as.Process *Coerce into a Process*

Description

This function converts objects into a process. If no meta data is provided it will return a valid user defined process, not yet storable in the back-end.

Usage

```
as.Process.ProcessInfo(from)

as.Process.Graph(from)

as.Process.ProcessNode(from)

as.Process.Service(from)

as.Process.function(from)

as.Process.Job(from)
```

Arguments

from	the source from which to coerce (ProcessInfo, Graph() or ProcessNode())

Value

Process()

BasicAuth *Basic Authentication class*

Description

This class handles the authentication to an openEO back-end that supports "basic" as login type. The class handles the retrieval of an access token by sending the encoded token consisting of user name and the password via HTTP header 'Authorization'. The authentication will be done once via login() or multiple times when the lease time runs out. This class is created and registered in the OpenEOClient(). After the login the user_id and the access_token are obtained and used as "bearer token" for the password restricted web services.

Details

The class inherits all fields and function from `IAuth()`

Value

an object of type `R6Class()` representing basic authentication

Methods

`$new(endpoint,user,password)` the constructor with the login endpoint and the credentials

Arguments

`endpoint` the basic authentication endpoint as absolute URL

`user` the user name

`password` the user password

`binary_ops` *Binary function wrappers*

Description

The functions here are used in combination with `ProcessGraphParameter` and `ProcessNode` in order to make it easier to write arithmetic functions for openEO user defined processes in R. The functions map into their openEO processes counterparts.

Usage

```
## S3 method for class 'ProcessNode'
e1 + e2

## S3 method for class 'ProcessGraphParameter'
e1 + e2

## S3 method for class 'ProcessNode'
e1 - e2

## S3 method for class 'ProcessGraphParameter'
e1 - e2

## S3 method for class 'ProcessNode'
e1 * e2

## S3 method for class 'ProcessGraphParameter'
e1 * e2
```

```
## S3 method for class 'ProcessNode'
e1 / e2

## S3 method for class 'ProcessGraphParameter'
e1 / e2

## S3 method for class 'ProcessNode'
e1 ^ e2

## S3 method for class 'ProcessGraphParameter'
e1 ^ e2

## S3 method for class 'ProcessNode'
e1 %% e2

## S3 method for class 'ProcessGraphParameter'
e1 %% e2

## S3 method for class 'ProcessNode'
e1 & e2

## S3 method for class 'ProcessGraphParameter'
e1 & e2

## S3 method for class 'ProcessNode'
e1 | e2

## S3 method for class 'ProcessGraphParameter'
e1 | e2

xor.ProcessNode(x, y)

xor.ProcessGraphParameter(x, y)

## S3 method for class 'ProcessNode'
e1 == e2

## S3 method for class 'ProcessGraphParameter'
e1 == e2

## S3 method for class 'ProcessNode'
e1 != e2

## S3 method for class 'ProcessGraphParameter'
e1 != e2

## S3 method for class 'ProcessNode'
e1 < e2
```

```
## S3 method for class 'ProcessGraphParameter'
e1 < e2

## S3 method for class 'ProcessNode'
e1 <= e2

## S3 method for class 'ProcessGraphParameter'
e1 <= e2

## S3 method for class 'ProcessNode'
e1 >= e2

## S3 method for class 'ProcessGraphParameter'
e1 >= e2

## S3 method for class 'ProcessNode'
e1 > e2

## S3 method for class 'ProcessGraphParameter'
e1 > e2
```

Arguments

e1	ProcessGraphParameter, ProcessNode or a list or vector, which internal data is passed into the function or a numeric value
e2	same as e1
x	the first expression in the xor statement
y	the second expression in the xor statement

Value

a ProcessNode

Boolean *Boolean*

Description

Inheriting from Argument() in order to represent a boolean / logical.

Value

Object of R6Class() representing a boolean / logical.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

BoundingBox *BoundingBox*

Description

Inheriting from Argument() in order to represent a bounding box / extent of an area of interest. Its value is usually a named list with "west","south","east" and "north". For this argument the 'bbox' object of the sf package is also recognized (sf::st_bbox()). This holds also true for classes that support sf::st_bbox() and return a valid 'bbox' object.

Value

Object of R6Class() representing a bounding box / extent.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

Examples

```
## Not run:
# most of the time BoundingBox is a choice as parameter value for
# spatial_extent in 'load_collection'
p = processes()

# using a list
bbox = list(west=10.711799440170706,
            east= 11.542794097651838,
            south=45.92724558214729,
            north= 46.176044942018734)

data = p$load_collection(id = "SENTINEL2_L2A",
                         spatial_extent = bbox,
                temporal_extent = list("2020-01-01T00:00:00Z", "2020-01-20T00:00:00Z"),
                         bands = list("B04","B08"))

# using sf bbox
```

```
bbox = st_bbox(c(xmin=10.711799440170706,
                 xmax= 11.542794097651838,
                 ymin=45.92724558214729,
                 ymax= 46.176044942018734),
               crs = 4326)

data = p$load_collection(id = "SENTINEL2_L2A",
                         spatial_extent = bbox,
                  temporal_extent = list("2020-01-01T00:00:00Z", "2020-01-20T00:00:00Z"),
                         bands = list("B04","B08"))

# objects supporting sf::st_bbox()
img = stars::read_stars(system.file("tif/L7_ETMs.tif",package = "stars"))
data = p$load_collection(id = "SENTINEL2_L2A",
                         spatial_extent = img,
                  temporal_extent = list("2020-01-01T00:00:00Z", "2020-01-20T00:00:00Z"),
                         bands = list("B04","B08"))


## End(Not run)
```

`capabilities` *Capabilities overview*

Description

The function queries the connected openEO service for general information about the service.

Usage

```
capabilities(con = NULL)
```

Arguments

con	A connected OpenEO client (optional), if omitted `active_connection()` is used

Value

capabilities object

client_version *Returns the client version*

Description

The function returns the client version. Wraps the call 'packageVersion("openeo")', which will return this packages version.

Usage

```
client_version()
```

Value

the client version

CollectionId *CollectionId class*

Description

Inheriting from `Argument()` in order to represent a CollectionId on an openeo back-end.

Value

Object of `R6Class()` representing a CollectionId.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

collection_viewer *View openEO collections*

Description

The function opens a viewer panel in RStudio which renders the collection information in an HTML. It reuses common components from the openeo-vue-components.

Usage

```
collection_viewer(x = NULL, con = NULL)
```

Arguments

x	(optional) character with the name of a collection or the `Collection` obtained with `describe_collection()`. If NULL is provided (default), the list of all collections is shown.
con	a specific connection (optional), last connected service if omitted.

compute_result *Executes a job and returns the data immediately*

Description

Executes a job directly on the connected openEO service and returns the data. During the execution phase the connection to the server remains open. This function allows to debug the code and check the results immediately. Please keep in mind, that computational functions might be related to monetary costs, if no 'free' plan is available. Make sure to keep the data selection relatively small, also some openEO service provider might offer limited processes support, e.g. not supporting UDFs at this endpoint. When a file format is set, then the process graph will be parsed and the arguments for 'save_result' will be replaced. If the 'stars' package is installed and parameter `as_stars` is set to TRUE, then the downloaded data is opened and interpreted into a stars object.

Usage

```
compute_result(
  graph,
  output_file = NULL,
  budget = NULL,
  plan = NULL,
  as_stars = FALSE,
  format = NULL,
  con = NULL,
  ...
)
```

Arguments

graph	a Graph(), a function returning a ProcessNode() as an endpoint or the ProcessNode() will return the results
output_file	storage location for the returned data
budget	numeric, maximum spendable amount for testing
plan	character, selection of a service plan
as_stars	logical to indicate if the data shall be interpreted as a stars object
format	character or FileFormat specifying the File format for the output, if 'save_result' is not set in the process then it will be added otherwise the value stated here will replace the original value.
con	connected and authenticated openEO client (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits') or additional arguments that shall be passed to the openEO process 'save_result'

Value

a local path to the downloaded file or a stars object if as_stars=TRUE

Note

If parameter 'format' is ignored, it is assumed that 'save_result' was already used in the process graph. Otherwise it is up to the back-end provider how data is stored if 'save_result' was omitted.

---

conformance *OGC conformance*

Description

Queries the openEO service for the conformance. As stated in the API it is highly optional and only available if the service wants to achieve full compatibility with OGC API clients. This function queries the /conformance endpoint and returns it results as a list object translated from JSON using the jsonlite package.

Usage

```
conformance(con = NULL)
```

Arguments

con	a connected openEO client object (optional) otherwise active_connection() is used.

connect *Connect to a openEO service*

Description

Connects to openEO service. If the back-end provides a well-known endpoint that allows redirecting to specific versions you should provide the version parameter.

Usage

```
connect(host, version = NULL, exchange_token = "access_token", ...)
```

Arguments

host	URL pointing to the openEO server service host
version	the openEO API version number as string (optional), see also api_versions()
exchange_token	'access_token' or 'id_token' defines in the OIDC case the bearer token use
...	parameters that are passed on to login()

Details

You can explore several already available openEO web services by using the openEO hub (https://hub.openeo.org/). There you have an overview about their status and connection details like the URL and supported features. You can explore the service for free through the access to publicly available metadata of data collections as well as the offered processing functions. For any computation and the creation of web services, you need to register the openEO partner of your choice. There you will get further information on credentials and the log in procedure.

The ... parameter allows you to pass on arguments directly for login(). If they are omitted the client will only connect to the back-end, but does not do authentication. The user must do that manually afterwards. Based on the provided login parameters user / password or OIDC provider the appropriate login procedure for basic authentication or OIDC authentication will be chosen.

The parameter version is not required. If the service offers a well-known document of the service the client will choose an appropriate version (default the most recent production ready version).

When calling this function the OpenEOClient() is also stored in a variable in the package which marks the latest service that was connected to.

See Also

active_connection()

Examples

```
## Not run:
# The following examples show different configuration settings and point
# to imaginary URLs. Please obtain a valid URL via the openEO hub and
# register with one of the provider if required.

# connect to a host of the latest version and without authentication
con = connect(host='http://example.openeo.org')

# connect to a host by direct URL and basic log in
con = connect(host='http://example.openeo.org/v1.0',
              user='user',
             password='password')

# connect to a host with open id connect authentication
con = connect(host='http://example.openeo.org')

# connect and login with a named and valid oidc provider
con = connect(host='http://example.openeo.org',
              provider='your_named_provider')

## End(Not run)
```

---

create_job *Creates a new job on the back-end*

Description

In preparation to execute the users analysis workflow (user defined process) asynchronously, they need to register a job that will be scheduled when the required resources are available. To do so the user provides the process graph with optional descriptive meta data and the desired execution plan or the maximum amount of credits spent.

Usage

```
create_job(
  graph = NULL,
  title = NULL,
  description = NULL,
  plan = NULL,
  budget = NULL,
  con = NULL,
  ...
)
```

Arguments

graph	A Graph(), a function returning a ProcessNode() as an endpoint or the ProcessNode() will return the results
title	Optional title of a job
description	Optional detailed information about a job
plan	An optional execution plan offered by the back-end, determining how the job will be executed
budget	An optional budget, which sets the maximum amount of credits to be used by the job
con	connected and authenticated openEO client (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

Value

the id of the job

---

create_service *Prepares and publishes a service on the back-end*

Description

The function will create a web service of a process graph / workflow on the connected openEO service.

Usage

```
create_service(
  type,
  graph,
  title = NULL,
  description = NULL,
  enabled = NULL,
  configuration = NULL,
  plan = NULL,
  budget = NULL,
  con = NULL,
  ...
)
```

Arguments

type	character - the OGC web service type name to be created or an object of type ServiceType obtainable through list_service_types()
graph	A Graph(), a function returning a ProcessNode() as an endpoint or the ProcessNode() will return the results
title	character (optional) - a title for the service intended for visualization purposes in clients
description	character (optional) - a short description of the service
enabled	logical - whether or not the service is active or not
configuration	a named list specifying the configuration parameter
plan	character - the billing plan
budget	numeric - the amount of credits that can be spent on this service
con	connected and authenticated openEO client object (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

Value

Service object

---

create_user_process *Stores a graph as user defined process on the back-end*

Description

Uploads the process graph information to the back-end and stores it. This can be used as a user defined process.

Usage

```
create_user_process(
  graph,
  id = NULL,
  summary = NULL,
  description = NULL,
  submit = TRUE,
  con = NULL,
  ...
)
```

Arguments

graph	a process graph definition
id	the title of the user process
summary	the summary for the user process (optional)
description	the description for the user process (optional)
submit	whether to create a new user process at the openEO service or to create it for local use (default set to submit = TRUE)
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

Details

The parameter submit will be deprecated in the future. Please use as(obj, "Process"). This function is useful when copying a JSON representation of your process graph to another software. In that case use udp = as(obj, "Process") and simply print or call object udp on the console.

Value

a list assembling a process graph description or the graph id if send

---

create_variable *Creates a variable in a process graph*

Description

This function creates a variable to be used in the designated process graph with additional optional information.

Usage

```
create_variable(
  name,
  description = NULL,
  type = NULL,
  subtype = NULL,
  default = NULL
)
```

Arguments

name	the name of the variable
description	an optional description of the variable
type	the type of the value that is replaced on runtime, default 'string'
subtype	the subtype of the type (as specified by openEO types)
default	the default value for this variable

Value

a ProcessGraphParameter() object

`Date` *Date*

Description

Inheriting from Argument() in order to represent a date.

Value

Object of R6Class() representing a date.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(),TemporalIntervals(), MetadataFilter()

`DateTime` *DateTime*

Description

Inheriting from Argument() in order to represent a date with time component.

Value

Object of R6Class() representing a date with time component.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(),TemporalIntervals(), MetadataFilter()

debug *Triggers debugging mode*

Description

The debugging mode is created to investigate the communication between server and client. The mode can be turned on or off, depending on the selected function (debug, debug.off). It is stored as an package internal environment and other package functions can access it naturally. By using the environment object, entries can be changed.

Usage

```
debug()

debug.off()

is.debugging()
```

delete_file *Delete a file from the user workspace*

Description

Sends a request to an openEO back-end in order to remove a specific file from the users workspaces.

Usage

```
delete_file(src, con = NULL)
```

Arguments

src	the relative file path of the source file on the openEO back-end that shall be deleted
con	authorized connection (optional) otherwise active_connection() is used.

Value

logical

delete_job *Delete a job*

Description

Deletes a job from the back-end.

Usage

```
delete_job(job, con = NULL)
```

Arguments

job	the job or the id of the job
con	authenticated Connection (optional) otherwise active_connection() is used.

Value

logical with state of success

delete_service *Deletes a service function for a job*

Description

Queries the back-end and removes the current set service function of job.

Usage

```
delete_service(service, con = NULL)
```

Arguments

service	the Service or its id
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.

delete_user_process *Deletes a user process*

Description

The function initiates the deletion of a user defined process on the back-end. Only the owning user can delete their process. The user defined process also should not be part of any particular job.

Usage

```
delete_user_process(id, con = NULL)
```

Arguments

id	the id of the user process
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.

describe_account *Get the current user account information*

Description

Calls endpoint /me to fetch the user account information of the user currently logged in.

Usage

```
describe_account(con = NULL)
```

Arguments

con	authenticated client object (optional) otherwise active_connection() is used.

Value

object of type user

describe_collection *Describe a collection*

Description

Queries an openEO back-end and retrieves a detailed description about one or more collections offered by the back-end.

Usage

```
describe_collection(collection = NA, con = NULL)
```

Arguments

collection	Collection object or the collections id
con	Authentication object (optional) otherwise active_connection() is used.

Value

a Collection object with detailed information about a collection.

describe_job *Fetches information about a job*

Description

Returns a detailed description about a specified job (e.g., the status)

Usage

```
describe_job(job, con = NULL)
```

Arguments

job	the job object or the id of the job
con	authenticated Connection (optional) otherwise active_connection() is used.

Value

a detailed description about the job

describe_process *Describe a process*

Description

Queries an openEO back-end and retrieves more detailed information about offered processes

Usage

```
describe_process(process = NA, con = NULL)
```

Arguments

process	id of a process to be described, the ProcessInfo object or a Process object
con	Authentication object (optional) otherwise active_connection() is used.

Value

a list of detailed information

describe_service *Describes a service*

Description

Queries the server and returns information about a particular service

Usage

```
describe_service(service, con = NULL)
```

Arguments

service	the Service object or its id
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.

Value

Service object

describe_user_process *Fetches the representation of a stored user defined process*

Description

The function queries the back-end for a specific user defined process and returns detailed information.

Usage

```
describe_user_process(id, con = NULL)
```

Arguments

id	The id of the user process on the back-end
con	connected and authenticated openEO client object (optional) otherwise active_connection() is used.

Value

the user process as a ProcessInfo class (list object)

dimensions *Returns dimension*

Description

Returns dimension

Usage

```
dimensions(x, ...)
```

Arguments

x	an object from which dimension information is returned
...	additional parameters to pass on to internal functions

Value

dimension information as list

dimensions.Collection *Returns dimension information*

Description

The function returns the dimension information of a Collection object. This object is usually obtained when calling describe_collection. It returns the meta data information for the cube dimensions.

Usage

```
## S3 method for class 'Collection'
dimensions(x, ...)
```

Arguments

x	a Collection object
...	parameters to pass on (not used)

Value

dimension information as list

disconnect *disconnect*

Description

Uses the connections disconnect method to logout and clear all variables in the package for this active back-end. This will also refresh RStudios connection observer if it can be found.

Usage

```
disconnect()
```

Value

invisible NULL

download_file *Download a file from the user workspace*

Description

Sends a request to an openEO back-end to access the users files and downloads them to a given location.

Usage

```
download_file(src, dst = NULL, con = NULL)
```

Arguments

src	the relative file path of the source file on the openEO back-end
dst	the destination file path on the local file system
con	authorized connection (optional) otherwise active_connection() is used.

Value

The file path of the stored file on your machine

download_results *Downloads the results of a job*

Description

The function will fetch the results of a asynchronous job and will download all files stated in the links. The parameter 'folder' is the target location on the local computer.

Usage

```
download_results(job, folder, con = NULL)
```

Arguments

job	job object or the job_id for which the results are fetched. Also the return value of list_results() or its 'assets' field is also accepted.
folder	a character string that is the target path on the local computer
con	a connected and authenticated openEO connection (optional) otherwise active_connection() is used.

Value

a list of the target file paths or NULL if 'job' was incorrect

EPSGCode *EPSGCode class*

Description

Inheriting from `Argument()` in order to represent an EPSG Code. Allowed values are single integer values like `4326` or a text containing 'EPSG:' like `EPSG:4326`.

Value

Object of `R6Class()` representing an EPSG code as Integer

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

estimate_job *Estimates job costs*

Description

Calls the back-end and asks for an approximation about the monetary costs, the required time, and whether or not the job owners data download is already included in the monetary costs.

Usage

```
estimate_job(job, con = NULL)
```

Arguments

job	the job or the id of the job
con	authenticated Connection (optional) otherwise `active_connection()` is used.

Value

JobCostsEstimation containing information how much money and time will be spent

GeoJson *GeoJson*

Description

Inheriting from `Argument()` in order to represent a GeoJson object. This class represents geospatial features. Allowed values are either a list directly convertible into a valid GeoJson or polygon features of type 'sf' or 'sfc' from package 'sf'. The current implementation follows the data representation of 'sf' - meaning that coordinate order is XY (e.g. if CRS84 is used then lon/lat is the default order).

Details

As GeoJSON is defined in RFC7946 the coordinate reference system is `urn:ogc:def:crs:OGC::CRS84`, which uses a longitude, latitude ordering of the coordinates.

Value

Object of `R6Class()` representing an object in GeoJson.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

get_sample *Get sample data*

Description

In order to inspect data locally a very small spatial extent will be processed, downloaded and made available in R.

Usage

```
get_sample(
  graph,
  replace_aoi = TRUE,
  spatial_extent = NULL,
  execution = "sync",
  immediate = TRUE,
  con = NULL,
  ...
)
```

Arguments

graph	a ProcessGraph, a Process or the final node in a process for which the sample shall be calculated
replace_aoi	a logical flag to indicate whether or not the original spatial extent shall be substituted with a different one, default TRUE
spatial_extent	a bounding box or a spatial feature from which to derive a bounding box
execution	sync or async which indicates the processing chain, a not "async" value results in a synchronous processing
immediate	flag to be considered if the retrieval shall be immediately queued on the back-end
con	connected and authenticated openEO client (optional) otherwise active_connection() is used.
...	additional parameters that are passed to compute_result() or create_job()

Details

In order to get a better understanding about the processing mechanisms and the data structures used in the openEO back-end, it helps to check the actual data from time to time. This function aids the user in doing to. It replaces all spatial extents of the derived process graph with a new spatial extent which is calculated by the first spatial extent of the mandatory openEO process 'load_collection'. We take the center of the extent and add 0.0003 degrees to it. In case the coordinate reference system is not in WGS84, then the bounding box will be transformed into geodetic WGS84 beforehand, if the package 'sf' is present.

If the spatial extent was explicitly set to a small custom extent, then you can disable the replacement of the area of interest with replace_aoi = FALSE.

Graph *Graph object*

Description

This class represents an openEO process graph - which is generally denoted as field process_graph in the exchange objects of the API. The graph consists of ProcessNode()s and optional ProcessGraphParameter() (former variables). The explicit creation of a Graph is usually not required and discouraged, because this will be handled automatically.

Details

In terms of the openEO API the process graph is the technical description of a process. To create a user-defined process it requires a process graph and additional meta data. The process graph is not accepted at any openEO endpoint directly. Therefore, it has to be wrapped in a Process() object. Use as.Process() in those cases. It is similarly handled in other functions of this package.

Value

Object of R6Class() with methods for building an openEO process graph

Fields

`data` a named list of collection ids or process graph parameters depending on the context

Methods

`$new(final_node=NULL)` The object creator created from processes and available data.

`$getNodes()` a function to return a list of created `ProcessNode()`s for this graph

`$serialize()` creates a list representation of the graph by recursively calling `$serialize`

`$validate()` runs through the nodes and checks the validity of its argument values

`$getNode(node_id)` searches and returns a node from within the graph referenced by its node id

`$addNode(node)` adds a `ProcessNode()` to the graph

`$removeNode(node_id)` removes a process node from the graph

`$getFinalNode()` gets the result process node of a process graph

`$setFinalNode(node)` sets the result process node by node id or a ProcessNode

`$getVariables()` creates a named list of the defined variables of a process graph

`$setVariables(list_of_vars)` sets the `ProcessGraphParameter()` (former variables) of graph

Arguments

`final_node` optional, the final node (end node) that was used to create a graph

`node_id` the id of a process node

`node` process node or its node id

`parameter` the name of a parameter in a process

`value` the value to be set for a parameter of a particular process

`id` **or** `variable_id` the variable id

`description` a description field for a variable

`type` the type of variable, default 'string'

`default` optional default value to be set for a variable

`graphToJSON-deprecated` **toJSON functions*

Description

Those functions serialized a Graph or Process object into JSON text. They are deprecated. Use `toJSON` instead.

Usage

```
graphToJSON(x, ...)

processToJSON(x, ...)
```

Arguments

x	Graph or Process object
...	arguments for jsonlite::toJSON

`group_ops` *Group operator wrappers*

Description

R's mathematical group primitives that are translated to openEO processes.

Usage

```
## S3 method for class 'ProcessNode'
sum(..., na.rm = FALSE)

## S3 method for class 'ProcessGraphParameter'
sum(..., na.rm = FALSE)

## S3 method for class 'list'
sum(..., na.rm = FALSE)

## S3 method for class 'ProcessNode'
prod(..., na.rm = TRUE)

## S3 method for class 'ProcessGraphParameter'
prod(..., na.rm = TRUE)

## S3 method for class 'list'
prod(..., na.rm = TRUE)

## S3 method for class 'ProcessNode'
min(..., na.rm = TRUE)

## S3 method for class 'ProcessGraphParameter'
min(..., na.rm = TRUE)

## S3 method for class 'list'
min(..., na.rm = TRUE)

## S3 method for class 'ProcessNode'
max(..., na.rm = TRUE)

## S3 method for class 'ProcessGraphParameter'
max(..., na.rm = TRUE)
```

```
## S3 method for class 'list'
max(..., na.rm = TRUE)

## S3 method for class 'ProcessNode'
range(..., na.rm = TRUE)

## S3 method for class 'ProcessGraphParameter'
range(..., na.rm = TRUE)

## S3 method for class 'list'
range(..., na.rm = TRUE)

## S3 method for class 'ProcessNode'
mean(x, na.rm = FALSE, ...)

## S3 method for class 'ProcessGraphParameter'
mean(x, na.rm = FALSE, ...)

## S3 method for class 'list'
mean(x, na.rm = FALSE, ...)

## S3 method for class 'ProcessNode'
median(x, na.rm = FALSE, ...)

## S3 method for class 'ProcessGraphParameter'
median(x, na.rm = FALSE, ...)

## S3 method for class 'list'
median(x, na.rm = FALSE, ...)

sd.ProcessNode(x, na.rm = FALSE)

sd.ProcessGraphParameter(x, na.rm = FALSE)

sd.list(x, na.rm = FALSE)

var.ProcessNode(x, na.rm = FALSE)

var.ProcessGraphParameter(x, na.rm = FALSE)

var.list(x, na.rm = FALSE)
```

Arguments

...	multiple arguments that start with a ProcessNode or a ProcessGraphParameter
na.rm	logical to determine if NA values shall be removed in the calculation
x	a vector or list of values that are mixed or consist fully of ProcessNode, ProcessGraphParameter or numerical values

Details

The ... parameter is required to start with the ProcessNode or a ProcessGraphParameter that returns a numeric value. If it starts with a number the corresponding function in base R will be used, which will result in most cases in an error because base R cannot interprete the ProcessNode and ProcessGraphParameter objects. In that case you need to reorder the elements so that openeo's group operators will be used.

Value

ProcessNode

IAuth *IAuth*

Description

An interface that states the intended behavior for the authentication.

Fields

`access_token` The access_token to query password restricted webservices of an openEO back-end

`id_token` The id_token retrieved when exchanging the access_token at the identity provider

Methods

`$login()` Initiates the authentication / login in order to obtain the access_token

`$logout()` Terminates the access_token session and logs out the user on the openEO back-end

See Also

BasicAuth(), OIDCAuth()

Integer *Integer class*

Description

Inheriting from Argument() in order to represent a single integer value.

Value

Object of R6Class() representing an Integer

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

JobId *JobId class*

Description

Inheriting from Argument() in order to represent a jobId on an openeo back-end.

Value

Object of R6Class() representing the id of a job.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

Kernel *Kernel*

Description

Inheriting from Argument() in order to represent a 2-dimensional array of weights applied to the x and y (spatial) dimensions of the data cube. The inner level of the nested array is aligned to the x-axis and the outer level is aligned to the y-axis. Each level of the kernel must have an uneven number of elements.

Value

Object of R6Class() representing a Kernel.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

list_collections *List data on connected server*

Description

List available collections stored on an openEO server and return them as a `CollectionList` - a named list of `Collection` objects. The names are the collection IDs. Although the result at `describe_collection()` is also a Collection, the Collection object of returned from `list_collections()` is considered a list entry with less detailed information. The collection list is stored internally as the package environment variable `data_collection`, which can be accessed and set with `active_data_collection()`.

Usage

```
list_collections(con = NULL)

active_data_collection(collection = NULL)
```

Arguments

`con`	Connection object (optional) otherwise `active_connection()` is used.
`collection`	the 'CollectionList' object of list_collections to be set as the active data collection in the package environment or left empty or NULL to return the package environment variable.

Value

object of class 'CollectionList'

list_features *List the openEO endpoints*

Description

The client queries the version resolved back-end for its endpoint capabilities and returns it as a tibble.

Usage

```
list_features(con = NULL)
```

Arguments

`con`	A connected openEO client (optional) otherwise `active_connection()` is used.

Value

`data.frame` or `tibble` (if available)

list_files *List workspace files*

Description

Lists all files in the workspaces of the authenticated user.

Usage

```
list_files(con = NULL)
```

Arguments

con	authorized connection (optional) otherwise active_connection() is used.

Value

a data.frame or tibble with file names and storage sizes

list_file_formats *Supported Input/Output formats*

Description

The function queries the openEO service for supported I/O formats as a FileFormatList object.

Usage

```
list_file_formats(con = NULL)
```

Arguments

con	openEO client object (optional) otherwise active_connection() is used.

Details

The FileFormatList object is a named list, which is organized into 'input' and 'output'. For each category a different named list with the FileFormat is indexed by its format ID.

Value

a FileFormatList object

list_jobs *List the jobs of a user*

Description

Lists the jobs that a user has uploaded or in that are in execution

Usage

```
list_jobs(con = NULL)
```

Arguments

con	the authenticated Connection (optional) otherwise active_connection() is used.

list_oidc_providers *Available OIDC provider*

Description

In case the openEO service provider supports OpenID connect authentication, this function will return a list of supported provider that can be used by this specific service.

Usage

```
list_oidc_providers(con = NULL)
```

Arguments

con	active openEO service connection (OpenEOClient())

Value

a ProviderList object which is a named list of Provider objects.

list_processes *List available processes on server*

Description

List all processes available on the back-end. This returns the R translation of the JSON metadata as lists. This process description is stored internally at the environment package variable `process_list`, which is not directly accessible apart from this function.

Usage

```
list_processes(con = NULL)
```

Arguments

con	Connection object (optional) otherwise `active_connection()` is used.

Value

a list of lists with process_id and description

list_results *Creates a list of download paths*

Description

The function queries the back-end to receive the URLs to the downloadable files of a particular job.

Usage

```
list_results(job, con = NULL)
```

Arguments

job	the job object or the id of the job
con	connected and authenticated openEO client object (optional) otherwise `active_connection()` is used.

Value

result object containing of URLs for download

list_services *Lists the current users services*

Description

Queries the back-end to retrieve a list of services that the current user owns. Services are web services like WCS, WFS, etc. The result is an object of type `ServiceList`, which is a named list of `Service`. The indices are the service IDs, the service object that is indexed by its ID and may use other functions instead of its service ID.

Usage

```
list_services(con = NULL)
```

Arguments

con	connected and authenticated openEO client object (optional) otherwise `active_connection()` is used.

Value

named list of Services (class `ServiceList`)

list_service_types *Returns the web service types of the back-end*

Description

The function queries the back-end for the supported web service types usable by the client and returns a named list of `ServiceType` indexed by the service type ID. ServiceTypes can be used when creating a supported web service from the user defined process (process graph).

Usage

```
list_service_types(con = NULL)
```

Arguments

con	a connected openEO client object (optional) otherwise `active_connection()` is used.

Value

a `ServiceTypeList`

list_udf_runtimes *Lists the supported UDF runtimes*

Description

The function queries the back-end for its supported UDF runtimes and returns detailed information about each runtime.

Usage

```
list_udf_runtimes(con = NULL)
```

Arguments

con	connected and authenticated openEO client object (optional) otherwise active_connection() is used.

Value

list of UDF runtimes with supported UDF types, versions and installed packages

list_user_processes *Lists the IDs of the process graphs from the current user.*

Description

Queries the back-end to retrieve a list of graph ids that the current user has stored on the back-end.

Usage

```
list_user_processes(con = NULL)
```

Arguments

con	connected and authenticated openEO client object (optional) otherwise active_connection() is used.

Value

a named list of user defined processes (ProcessInfo)

`login` *Log in on a specific back-end*

Description

Retrieves the bearer-token from the back-end by sending user name and password to the back-end. This step is usually performed during the 'connect' step. If you are only connected to a back-end in order to explore the capabilities and want to compute something, then you need to log in afterwards.

Usage

```
login(user = NULL, password = NULL, provider = NULL, config = NULL, con = NULL)
```

Arguments

`user`	the user name
`password`	the password
`provider`	provider object as obtained by 'list_oidc_providers()' or the name of the provider in the provider list. If NULL and `provider_type="oidc"` then the first available provider is chosen from the list.
`config`	named list containing 'client_id' and 'secret' or a path to the configuration file (type JSON). If NULL and `provider_type="oidc"` the configuration parameters are taken from the default authentication client of the OIDC provider.
`con`	connected back-end connection (optional) otherwise `active_connection()` is used.

Details

Based on the general login type (BasicAuth or OIDCAuth) there need to be different configurations. The basic authentication (if supported) is the simplest login mechanism for which user need to enter their credentials directly as `user` and `password`.

For the Open ID connect authentication the user needs to select one of the accepted OIDC providers of `list_oidc_providers()` as `provider`. Alternatively the name of the provider suffices. For further configuration, you can pass a named list of values as `config` or a file path to a JSON file.

There are many different authentication mechanisms for OIDC and OAuth2.0, which OIDC is based on. The 'openeo' package supports currently the authorization_code, authorization_code+pkce, device_code and device_code+pkce (see OIDCAuth). For authorization_code you need to state the `client_id` and `secret` in the configuration options. In general the most comfortable available login mechanism is chosen automatically (1. device_code+pkce, 2. device_code 3. authorization_code+pkce, 4. authorization_code). For example, with the device_code flow you normally don't even need to specify any additional configuration.

If you really want to choose the authorization flow mechanism manually, you can add `grant_type` in the configuration list. You can then use the following values:

- authorization_code

- authorization_code+pkce
- urn:ietf:params:oauth:grant-type:device_code
- urn:ietf:params:oauth:grant-type:device_code+pkce

Value

a connected and authenticated back-end connection

Configuration options

`client_id` The client id to use, when authorization code is selected as grant_type

`secret` The client secret that matches the client_id to identify and validate this local client towards the identity provider

`grant_type` Manually selected authentication method from the ones stated above.

`scope` Manually select the scopes for the authentication method. Note: this is usually filled automatically with the information from the provider object

Examples

```
## Not run:
# simple connection without login to maybe explore the capabilities of a back-end first
# the URL won't work and is just to demonstrate how to write the code
con = connect(host='http://example.openeo.org',version='1.0.0')

# some back-ends support logging in throug OIDC without any parameters
login()

# basic authentication, credentials are dummy values
login(user='user',password='password')

# or alternatively the OIDC login
login(provider=provider, config=config)

# with device_code+pkce enabled at the OIDC provider you can even use this
login(provider="your_named_provider")


## End(Not run)
```

---

logout *Log out*

Description

Logs out or closes the active connection to an openEO service.

Usage

```
logout(con = NULL)
```

Arguments

con	a connected openEO client object (optional) otherwise active_connection() is used.

`logs` *Access logs of a Service or Job*

Description

Prints contents of the log file of a Job or Service to the console. Requests the log every second if the service is enabled or the batch job is active. If the log response always empty for a given timeout, the logging stops. Also if the job or service is not active at the moment timeout is ignored and the log is just printed once. To call the different logs log_job() or log_service() are used internally.

Usage

```
logs(obj = NULL, job_id = NULL, service_id = NULL, con = NULL, timeout = NULL)
```

Arguments

obj	Service or Job object
job_id	character the jobs ID
service_id	character - the services ID
con	a connected openEO client (optional) otherwise active_connection() is used.
timeout	integer the timeout for the logging of active jobs or services after no update in seconds, if omitted it is determined internally (running / queued / enabled -> 60s)

Details

In Jupyter, RMarkdown and knitr HTML environments the timeout parameter does not apply and this function only returns the logs that are available at the time of the request. To refresh the logs, you have to re-execute the function again.

See Also

log_job() or log_service()

log_job *Job log*

Description

Opens the log of job.

Usage

```
log_job(job, offset = NULL, limit = NULL, con = NULL)
```

Arguments

job	the job or the job_id
offset	the id of the log entry to start from
limit	the limit of lines to be shown
con	an optional connection if you want to address a specific service

Value

a Log object

log_service *Service log*

Description

Opens the log of secondary service.

Usage

```
log_service(service, offset = NULL, limit = NULL, con = NULL)
```

Arguments

service	the service or the service_id
offset	the id of the log entry to start from
limit	the limit of lines to be shown
con	an optional connection if you want to address a specific service

Value

a Log object

MetadataFilter *MetadataFilter*

Description

Inheriting from ProcessGraphArgument() in order to represent a list of functions that is internally interpreted into Process() objects.

Value

Object of R6Class() representing a list of Process() in order to filter for collections.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(),TemporalIntervals(), MetadataFilter()

Examples

```
## Not run:
# define filter statement
filter = list(
   "eo:cloud_cover" = function(x) x >= 0 & x < 50,
   "platform" = function(x) x == "Sentinel-2A"
)

# setting the arguments is done via the process graph building with of 'processes()'

## End(Not run)
```

Number *Number class*

Description

Inheriting from Argument() in order to represent a numeric value.

Value

Object of R6Class() representing a number

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

OIDCAuth *OIDC Authentication*

Description

defines classes for different OpenID connect interaction mechanisms. The classes are modeled in generalized fashion by inheriting functions from IAuth and AbstractOIDCAuthentication.

Details

The openEO conformant back-ends shall offer either a basic authentication and / or an OpenID Connect (OIDC) authentication. The first is covered at BasicAuth. And since OIDC is based on the OAuth2.0 protocol there are several mechanisms defined to interact with an OIDC provider. The OIDC provider can be the back-end provider themselves, but they can also delegate the user management to other platforms like EGI, Github, Google, etc, by pointing to the respective endpoints during the service discovery of the back-end. Normally users would not create those classes manually, but state the general login type (oidc or basic) and some additional information (see login).

This client supports the following interaction mechanisms (grant types):

- authorization_code
- authorization_code+pkce
- urn:ietf:params:oauth:grant-type:device_code+pkce

authorization_code: During the login process an internet browser window will be opened and you will be asked to enter your credentials. The website belongs to the OIDC provider of the chosen openEO back-end. Meanwhile, the client will start a server daemon in the background that listens to the callback from the OIDC provider. For this to work the user needs to get in contact with the openEO service provider and ask them for a configuration file that will contain information about the client_id and secret. The redirect URL requested from the provider is http://localhost:1410/

authorization_code+pkce: This procedure also spawns a temporary web server to capture the redirect URL from the OIDC provider. The benefit of this mechanism is that it does not require a client secret issued from the OIDC provider anymore. However, it will still open the internet browser and asks the user for credentials and authorization.

device_code+pkce: This mechanism does not need to spawn a web server anymore. It will poll the endpoint of the OIDC provider until the user enters a specific device code that will be printed onto the R console. To enter the code either the URL is printed also to the console or if R runs in the interactive mode the internet browser will be opened automatically.

device_code: This mechanism uses a designated device code for human confirmation. It is closely related to the device_code+pkce code flow, but without the additional PKCE negotiation.

Fields

`access_token` The access_token to query password restricted webservices of an openEO back-end

`id_token` The id_token retrieved when exchanging the access_token at the identity provider

Methods

`$new(provider, config=NULL, ...)` the constructor for the authentication

`$login()` Initiates the authentication / login in order to obtain the access_token

`$logout()` Terminates the access_token session and logs out the user on the openEO back-end

`$getUserData()` queries the OIDC provider for the user data like the 'user_id'

`$getAuth()` returns the internal authentication client as created from package 'httr2'

Arguments

`provider` the name of an OIDC provider registered on the back-end or a provider object as returned by `list_oidc_providers()`

`config` either a JSON file containing information about 'client_id' and 'secret' or a named list. Experienced user and developer can also add 'scopes' to overwrite the default settings of the OIDC provider

`...` additional parameter might contain `force=TRUE` specifying to force the use of a specific authentication flow

See Also

openEO definition on Open ID connect https://openeo.org/documentation/1.0/authentication.html#openid-connect

Open ID Connect (OIDC) https://openid.net/connect/

OAuth 2.0 Device Authorization Grant https://datatracker.ietf.org/doc/html/rfc8628

Proof Key for Code Exchange by OAuth Public Clients https://datatracker.ietf.org/doc/html/rfc7636

`openeo-deprecated` *openeo-deprecated*

Description

Lists all currently deprecated functions that will be removed in the future.

Deprecated

graphToJSON(x,...) replaced by toJSON

processToJSON(x,...) replaced by toJSON

`OpenEOClient` *openEO client class*

Description

An R6Class that interacts with an openEO compliant back-end.

Fields

`user_id` The user_id obtained after authentication

`api.mapping` The mapping of the API endpoints and the back-end published ones

Methods

`$new(host=NULL)` the constructor with an optional host URL to connect to

`$getBackendEndpoint(endpoint_name)` returns the URL for the requested endpoint tag

`$request(tag,parameters=NULL,authorized=FALSE, ...)` performs the desired HTTP request by endpoint tag with path parameters and whether or not authorization (access_token) is necessary

`$isConnected()` whether or not the client has a host set

`$isLoggedIn()` returns a logical describing whether the user is logged in

`$getHost()` returns the host URL

`$stopIfNotConnected()` throws an error if called and the client is not connected

`$connect(url=NULL,version=NULL)` connects to a specific version of a back-end

`$disconnect()` disconnects from the back-end by logout and clearing of active back-end package variables

`$api_version()` returns the openEO API version this client is compliant to

`$login(user=NULL, password=NULL,provider=NULL,config=NULL)` creates an `IAuth()` object

`$logout()` invalidates the access_token and terminates the current session

`$getAuthClient()` returns the authentication client

`$setAuthClient(value)` sets the authentication client if it was configured and set externally

`$getCapabilities()` service exploration to retrieve the supported openEO endpoints

`$getId()` returns the ID of the Connection as stated in the getCapabilities document

`$getTitle()` returns the title of the connection as stated in the getCapabilities document

Arguments

`host` the openEO host URL

`endpoint_name` the endpoint tag the client uses for the endpoints

`tag` endpoint tag

`parameters` named list of values to be replaced in the endpoint

`authorized` whether or not the endpoint requires authentication via access_token

`url` url of an openEO back-end either directly versioned or with the separate version statement

`version` the openEO API version to be used, or a list of available API versions if set to NULL

`user` the user name

`password` the user password

`value` an authentication object

`OutputFormat` *OutputFormat class*

Description

Inheriting from `Argument()` in order to represent an output format of a back-end as a character string value.

Value

Object of `R6Class()` representing an output format of a back-end.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

`OutputFormatOptions` *OutputFormatOptions*

Description

Inheriting from `Argument()` in order to represent the additional output format options of a back-end.

Value

Object of `R6Class()` representing output format options.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

`Parameter` *Parameter class*

Description

This class defines parameters of `Process()`. They store information about the type, format and pattern. A parameter class is designed to not carry any value, as opposed to an `Argument()`.

Details

The parameters are parsed from the specific description and format of the JSON objects returned for the parameters in processes. Find a list of openEO-specific formats here: RFC7946

Value

Object of `R6Class()` which represents a parameter.

Methods

`$new(name, description, required=FALSE)`

`$getName` returns the name of a parameter as string

`$setName(name)` sets the name of a parameter

`$getDescription()` returns the description of a parameter

`$setDescription(description)` sets the description of a parameter

`$getPattern()` returns a string with the pattern of a parameter description

`$setPattern(pattern)` sets the pattern (string) for a parameter

`$getDefault()` returns the parameter's default value

`$setDefault(default)` sets the default value of a parameter

`$matchesSchema(schema)` returns TRUE if the given schema - a list of the parsed openEO API schema object - matches the parameter's schema, which is used for finding the corresponding parameter

`$getSchema()` returns the schema definition

`$asParameterInfo()` returns a list representation of this parameter for being sent in a JSON to the openEO service

`$isNullable()` returns TRUE if the parameter is allowed to be nullable, FALSE otherwise

`$isRequired()` returns whether a parameter is mandatory or not

`$isAny()` returns TRUE if this parameter describes a choice of parameters

Arguments

`name` character - The name of a parameter
`description` character - The description of a parameter
`required` logical - whether it is required or not
`pattern` the regexp as a string indicating how to formulate the value
`default` the regexp as a string indicating how to formulate the value
`schema` the parsed schema object of a process parameter as a list

---

parse_graph *Converts a JSON openEO graph into an R graph*

Description

The function reads and parses a json text and creates a Graph object.

Usage

```
parse_graph(json, parameters = NULL, con = NULL)
```

Arguments

`json`	the json graph in a textual representation or an already parsed list object
`parameters`	optional parameters
`con`	a connected openEO client (optional) otherwise `active_connection()` is used.

Value

Graph object

---

print.ProcessInfo *Print an openEO process*

Description

Print function to visualize relevant information about an openEO process

Usage

```
## S3 method for class 'ProcessInfo'
print(x, ...)
```

Arguments

`x`	process info that is received on list_processes and describe_process
`...`	additional parameters (not used)

print.User *Prints a User object*

Description

A visualization for the user account information obtained by /me

Usage

```
## S3 method for class 'User'
print(x, ...)
```

Arguments

x	an User object that can be retrieved at describe_account
...	additional parameters (not used)

privacy_policy *Visualize the privacy policy*

Description

If the service provides information about their privacy policy in their capabilities, the function opens a browser window to visualize the web page.

Usage

```
privacy_policy(con = NULL)
```

Arguments

con	a connected openEO client object (optional) otherwise active_connection() is used.

Value

a list of the link identifying the privacy policy from the service capabilities or NULL

`Process` *Process object*

Description

This object reflects a process offered by an openEO service in order to load and manipulate data collections. It will be created with the information of a received JSON object for a single process, after the arguments of the process have been translated into `Argument()` objects.

Value

Object of `R6Class()` with methods for storing meta data of back-end processes and user assigned data

Fields

`parameters` • a named list of Argument objects

`isUserDefined` logical - depending if the process is offered by the openEO service or if it was user defined

Methods

$new(id,parameters,description=character(), summary = character(), parameter_order=character(),returns)

$getId() returns the id of a process which was defined on the back-end

$getParameters() returns a named list of arguments

$getReturns() returns the schema for the return type as list

$getFormals() returns the function formals for this process - usually a named vector of the specified default values, but NA where no default value was specified

$setId(id) sets the id of a process

$setSummary(summary) sets the summary text

$setDescription(description) sets the description text

$getParameter(name) returns the Argument object with the provided name

$getProcessGraph() returns the ProcessGraph to which this Process belongs

$setProcessGraph(process_graph) sets the ProcessGraph to which this Process belongs

$validate() validates the processes argument values

$serialize() serializes the process - mainly used as primary serialization for a `ProcessNode()`

$getCharacteristics() select all non functions of the private area, to be used when copying process information into a process node

Arguments

id process id from the back-end

parameters a list of Argument objects

description the process description

summary the summary of a process

returns the returns part of the process definition or an already evaluated parameter

name a parameter name

value the value for a parameter or the description text

ProcessCollection *Process Collection*

Description

This object contains template functions for process graph building from the processes offered by an openEO service. This object is an unlocked R6 object, in order to add new functions at runtime.

Methods

`$new(con = NULL)` The object creator created an openEO connection.

Arguments

con optional an active and authenticated Connection (optional) otherwise `active_connection()` is used.

See Also

`processes()`

---

processes *Get a process graph builder / process collection from the connection*

Description

Queries the connected back-end for all available processes and collection names and registers them via R functions on a `ProcessCollection` object to build a process graph in R. The current `ProcessCollection` is stored internally at the package environment variable `process_collection`, which can be fetched and set with `active_process_collection`.

Usage

```
processes(con = NULL)

active_process_collection(processes = NULL)
```

Arguments

con	a connection to an openEO back-end (optional) otherwise active_connection() is used.
processes	the ProcessCollection that is obtained from processes() to be set as the active process collection or left empty to fetch the ProcessCollection from the package variable.

Value

a ProcessCollection object with the offered processes of the back-end

---

ProcessGraphArgument *ProcessGraphArgument*

Description

Inheriting from Argument() in order to represent an argument that contains a process or a derivable value (formerly known as callback). The ProcessGraphArgument operates on the reduced data of a data cube. For example reducing or aggregating over the temporal dimension results in a time series that has to be reduced into a single value or aggregated into another time series. The value of a ProcessGraphArgument is usually a function that will be coerced into Process(). The function is required to use the same amount of parameters as ProcessGraphParameter objects are defined, because during the coercion those ProcessGraphParameter are passed to function. Additional information can be found in the openEO API documentation:

- https://api.openeo.org/#section/Processes/Process-Graphs

Value

Object of R6Class() representing a ProcessGraph.

Methods

$getProcessGraphParameters() returns the available list ProcessGraphParameter()

$setProcessGraphParameters(parameters) assigns a list of ProcessGraphParameter() to the ProcessGraph

Arguments

parameters the ProcessGraphParameter() list

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(),TemporalIntervals(), MetadataFilter()

ProcessGraphId *ProcessGraphId*

Description

Inheriting from `Argument()` in order to represent a process graph Id on an openeo back-end.

Value

Object of `R6Class()` representing the id of a process graph.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

ProcessGraphParameter *ProcessGraphParameter*

Description

Inheriting from `Argument()` in order to represent the available data within a ProcessGraph graph. Additional information can be found in the openEO API documentation:

- https://api.openeo.org/#section/Processes/Process-Graphs

Value

Object of `R6Class()` representing a ProcessGraph value.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

ProcessNode *Process Node object*

Description

This class inherits all functions and fields from `Process()` and extends it with a node id and a special serialization function. The ProcessNode is an essential building block of the `Graph()`.

Methods

$getNodeId() returns the node id

$setNodeId(id) set the node id, which is of interest when `parse_graph()` is executed

$serializeAsReference() during the serialization the process node might be used as a reference and this function serializes the process node accordingly

Arguments

id the node id

process_viewer *Viewer panel for provided openEO processes*

Description

Opens up a viewer panel in RStudio and renders one or more processes of the connected openEO service in HTML. The components of openeo-vue-components are reused.

Usage

```
process_viewer(x = NULL, con = NULL)
```

Arguments

x	(optional) a function from the `ProcessCollection()`, a `ProcessNode()`, `Process()` or a character containing the process id. If NULL is provided (default), the list of processes is shown.
con	a specific connection (optional), last connected service if omitted.

ProjDefinition *ProjDefinition*

Description

Inheriting from Argument() in order to represent a projection definition as a PROJ string.

Value

Object of R6Class() representing a projection definition based on PROJ.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(),TemporalIntervals(), MetadataFilter()

RasterCube *RasterCube*

Description

Inheriting from Argument() in order to represent a raster cube. This is usually the in- and output format of a process unless the process operates within a ProcessGraph on reduced data. The VectorCube() behaves comparably, but with underlying spatial feature data.

Value

Object of R6Class() representing a raster cube.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(),TemporalIntervals(), MetadataFilter()

remove_variable *Removes a variable from the Graph*

Description

The function that removes a selected variable from the graph. It is removed from the list of defined variables that are obtainable with `variables()`. The variables already placed in the graph won't be deleted, only in the defined variables list.

Usage

```
remove_variable(graph, variable)
```

Arguments

graph	a `Graph()` object
variable	a variable id or a variable object

Value

TRUE

send_udf *Test a UDF operation*

Description

This function is still under development and depends heavily on test data in a specific format and whether or not the back-end provider exposes their UDF service endpoint or if you have setup a local UDF service (see notes). The openEO UDF API v0.1.0 had foreseen to ship data and code in a single message and to be interpretable by a computing service a specific format was designed. Usually this whole operation is neatly hidden within the back-end, but if you want to test and debug the code, you need to create such data first. Some examples are available at https://github.com/Open-EO/openeo-r-udf/tree/master/examples/data.

Usage

```
send_udf(
  data,
  code,
  host = "http://localhost",
  port = NULL,
  language = "R",
  debug = FALSE,
  user_context = NA,
```

```
  server_context = NA,
  download_info = FALSE,
  legacy = FALSE,
  ...
)
```

Arguments

data	file path or a list object with the UDF-API data object
code	a call object or a file path of the user defined code
host	URL to the UDF service
port	(optional) port of the UDF service host
language	programming language (R or Python) of the source code
debug	(optional) logical - Switch on / off debugging information
user_context	list - Context parameter that are shipped from the user into the udf_service
server_context	list - Context usually sent from the back-end to trigger certain settings
download_info	(optional) logical - Whether or not to print the time taken separately for the download
legacy	logical - Whether or not the legacy endpoint is used (default: FALSE)
...	parameters passed on to httr::content or to be more precise to jsonlite::fromJSON

Details

Hint: If you use a local R UDF service you might want to debug using the 'browser()' function.

Value

the textual JSON representation of the result

Note

The debug options are only available for the R-UDF service. The R UDF-API version has to be of version 0.1.0 (not the old alpha version). You might want to check https://github.com/Open-EO/openeo-r-udf#running-the-api-locally for setting up a local service for debugging.

Examples

```
## Not run:
port = 5555
host = "http://localhost"
script = quote({
  all_dim = names(dim(data))
  ndvi_result = st_apply(data, FUN = function(X,...) {
    (X[8]-X[4])/(X[8]+X[4])
  }, MARGIN = all_dim[-which(all_dim=="band")])

  all_dim = names(dim(ndvi_result))
```

```
  min_ndvi = st_apply(ndvi_result,FUN = min, MARGIN = all_dim[-which(all_dim=="t")])

  min_ndvi
})
result = send_udf(data = "hypercube.json",code = script,host=host,port=port)


## End(Not run)
```

`start_job` *Starts remote asynchronous evaluation of a job*

Description

The function sends a start signal to the back-end triggering a defined job.

Usage

```
start_job(job, log = FALSE, con = NULL)
```

Arguments

`job`	the job object or the job id
`log`	logical - whether to enable automatic logging after starting the job
`con`	connected and authenticated openEO client (optional) otherwise `active_connection()` is used.

Value

the job object of the now started job

`status` *Retrieves the status*

Description

The function refreshes the passed object and returns its status.

Usage

```
status(x, ...)

## S3 method for class 'OpenEOClient'
status(x, ...)

## S3 method for class 'Job'
status(x, ...)

## S3 method for class 'Service'
status(x, ...)
```

Arguments

x	an object like Job
...	currently not used

Value

status as character

---

stop_job *Terminates a running job*

Description

Informs the server that the specified job needs to be terminated to prevent further costs.

Usage

```
stop_job(job, con = NULL)
```

Arguments

job	the job object or the id of job that will be canceled
con	authenticated Connection (optional) otherwise active_connection() is used.

Value

a success / failure notification

String *String class*

Description

Inheriting from `Argument()` in order to represent a character string value.

Value

Object of `R6Class()` representing a string.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

st_bbox.ProcessNode *st_bbox for ProcessNode*

Description

Traverses the graph from end node to roots and searches for defined bounding boxes in load_collection, filter_spatial, filter_bbox.

Usage

```
## S3 method for class 'ProcessNode'
st_bbox(obj, ...)
```

Arguments

`obj`	the process node
`...`	not used

Value

sf bbox object if one element was found, else a list of all bounding boxes (usually returned in EPSG:4326)

supports *Tag support lookup*

Description

Finds the client tag for a particular endpoint on the back-end and returns whether it is available or not.

Usage

```
supports(con = NULL, tag_name)
```

Arguments

con	backend connection (optional) otherwise active_connection() is used.
tag_name	the endpoints 'tag' name as character

Value

logical - whether the back-end supports the endpoint or not

TemporalInterval *TemporalInterval*

Description

Inheriting from Argument() in order to represent a temporal interval. Open interval borders are denoted by NA. Exactly two objects form the temporal interval.

Value

Object of R6Class() representing a temporal interval.

See Also

Array(), Integer(), EPSGCode(), String(), Number(), Date(), RasterCube(), VectorCube(), ProcessGraphArgument(), ProcessGraphParameter(), OutputFormatOptions(), GeoJson(), Boolean(), DateTime(), Time(), BoundingBox(), Kernel(), TemporalInterval(), TemporalIntervals(), CollectionId(), OutputFormat(), AnyOf(), ProjDefinition(), UdfCodeArgument(), UdfRuntimeArgument() and UdfRuntimeVersionArgument(), TemporalIntervals(), MetadataFilter()

TemporalIntervals *TemporalIntervals*

Description

Inheriting from `Argument()` in order to represent a list of `TemporalInterval()`.

Value

Object of `R6Class()` representing a list of temporal intervals.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

terms_of_service *Visualize the terms of service*

Description

If the service provides information about their terms of service in the capabilities, the function opens a new RStudio viewer panel and visualizes the HTML content of the link.

Usage

```
terms_of_service(con = NULL)
```

Arguments

con	a connected openEO client object (optional) otherwise `active_connection()` is used.

Value

a list of the link identifying the terms of service from the service capabilities or NULL

Time *Time*

Description

Inheriting from `Argument()` in order to represent the time of a day.

Value

Object of `R6Class()` representing the time of a day.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

toJSON *Wrapper for toJSON*

Description

This function is intended to have a preconfigured toJSON function to allow a user to visualize a process or graph in JSON. The JSON representation of a process is the same as it will be sent to the back-end.

Usage

```
## S4 method for signature 'Process'
toJSON(
  x,
  dataframe = c("rows", "columns", "values"),
  matrix = c("rowmajor", "columnmajor"),
  Date = c("ISO8601", "epoch"),
  POSIXt = c("string", "ISO8601", "epoch", "mongo"),
  factor = c("string", "integer"),
  complex = c("string", "list"),
  raw = c("base64", "hex", "mongo", "int", "js"),
  null = c("list", "null"),
  na = c("null", "string"),
  auto_unbox = FALSE,
  digits = 4,
  pretty = FALSE,
```

```
  force = FALSE,
  ...
)

## S4 method for signature 'Graph'
toJSON(
  x,
  dataframe = c("rows", "columns", "values"),
  matrix = c("rowmajor", "columnmajor"),
  Date = c("ISO8601", "epoch"),
  POSIXt = c("string", "ISO8601", "epoch", "mongo"),
  factor = c("string", "integer"),
  complex = c("string", "list"),
  raw = c("base64", "hex", "mongo", "int", "js"),
  null = c("list", "null"),
  na = c("null", "string"),
  auto_unbox = FALSE,
  digits = 4,
  pretty = FALSE,
  force = FALSE,
  ...
)
```

Arguments

x	a Process or Graph object
dataframe	how to encode data.frame objects: must be one of 'rows', 'columns' or 'values'
matrix	how to encode matrices and higher dimensional arrays: must be one of 'rowmajor' or 'columnmajor'.
Date	how to encode Date objects: must be one of 'ISO8601' or 'epoch'
POSIXt	how to encode POSIXt (datetime) objects: must be one of 'string', 'ISO8601', 'epoch' or 'mongo'
factor	how to encode factor objects: must be one of 'string' or 'integer'
complex	how to encode complex numbers: must be one of 'string' or 'list'
raw	how to encode raw objects: must be one of 'base64', 'hex' or 'mongo'
null	how to encode NULL values within a list: must be one of 'null' or 'list'
na	how to print NA values: must be one of 'null' or 'string'. Defaults are class specific
auto_unbox	automatically unbox() all atomic vectors of length 1. It is usually safer to avoid this and instead use the unbox() function to unbox individual elements. An exception is that objects of class AsIs (i.e. wrapped in I()) are not automatically unboxed. This is a way to mark single values as length-1 arrays.
digits	max number of decimal digits to print for numeric values. Use I() to specify significant digits. Use NA for max precision.
pretty	adds indentation whitespace to JSON output. Can be TRUE/FALSE or a number specifying the number of spaces to indent. See prettify()

force	unclass/skip objects of classes with no defined JSON mapping
...	additional parameters that are passed to jsonlite::toJSON

Value

JSON string of the process as a character string

Examples

```
## Not run:
# node is a defined process node
process = as(node, "Process")
toJSON(process)

graph = process$getProcessGraph()
toJSON(graph)

## End(Not run)
```

UdfCodeArgument *UdfCodeArgument class*

Description

Inheriting from `Argument()` in order to represent the UDF code that will be executed in a UDF call. The script has to be passed as a character string or as a local file path from which the script can be loaded.

Value

Object of `R6Class()` is an argument that expects an UDF code or a file path.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`, `TemporalIntervals()`, `MetadataFilter()`

UdfRuntimeArgument *UdfRuntimeArgument class*

Description

Inheriting from `Argument()` in order to represent the id of an UDF runtime object as obtainable by `list_udf_runtimes()`.

Value

Object of `R6Class()` representing the UDF runtime in a process argument.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

UdfRuntimeVersionArgument *UdfRuntimeVersionArgument class*

Description

Inheriting from `Argument()` in order to represent the id of a UDF runtime object as obtainable by `list_udf_runtimes()`.

Value

Object of `R6Class()` is an argument that expects a UDF runtime version or character as value.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

unary_ops *Unary function wrappers*

Description

The functions here are used in combination with `ProcessGraphParameter` and `ProcessNode` and facilitate writing arithmetic functions for openEO user defined processes in R. The functions translate into their openEO processes counterparts.

Usage

```
## S3 method for class 'ProcessNode'
abs(x)

## S3 method for class 'ProcessGraphParameter'
abs(x)

## S3 method for class 'ProcessNode'
sign(x)

## S3 method for class 'ProcessGraphParameter'
sign(x)

## S3 method for class 'ProcessNode'
sqrt(x)

## S3 method for class 'ProcessGraphParameter'
sqrt(x)

## S3 method for class 'ProcessNode'
trunc(x, ...)

## S3 method for class 'ProcessGraphParameter'
trunc(x, ...)

## S3 method for class 'ProcessNode'
floor(x)

## S3 method for class 'ProcessGraphParameter'
floor(x)

## S3 method for class 'ProcessNode'
ceiling(x)

## S3 method for class 'ProcessGraphParameter'
ceiling(x)
```

```
## S3 method for class 'ProcessNode'
round(x, digits = 0)

## S3 method for class 'ProcessGraphParameter'
round(x, digits = 0)

## S3 method for class 'ProcessNode'
exp(x)

## S3 method for class 'ProcessGraphParameter'
exp(x)

## S3 method for class 'ProcessNode'
log(x, base = exp(1))

## S3 method for class 'ProcessGraphParameter'
log(x, base = exp(1))

## S3 method for class 'ProcessNode'
log10(x)

## S3 method for class 'ProcessGraphParameter'
log10(x)

## S3 method for class 'ProcessNode'
cos(x)

## S3 method for class 'ProcessGraphParameter'
cos(x)

## S3 method for class 'ProcessNode'
sin(x)

## S3 method for class 'ProcessGraphParameter'
sin(x)

## S3 method for class 'ProcessNode'
tan(x)

## S3 method for class 'ProcessGraphParameter'
tan(x)

## S3 method for class 'ProcessNode'
cosh(x)

## S3 method for class 'ProcessGraphParameter'
cosh(x)
```

```
## S3 method for class 'ProcessNode'
sinh(x)

## S3 method for class 'ProcessGraphParameter'
sinh(x)

## S3 method for class 'ProcessNode'
tanh(x)

## S3 method for class 'ProcessGraphParameter'
tanh(x)

## S3 method for class 'ProcessNode'
acos(x)

## S3 method for class 'ProcessGraphParameter'
acos(x)

## S3 method for class 'ProcessNode'
asin(x)

## S3 method for class 'ProcessGraphParameter'
asin(x)

## S3 method for class 'ProcessNode'
atan(x)

## S3 method for class 'ProcessGraphParameter'
atan(x)

## S3 method for class 'ProcessNode'
acosh(x)

## S3 method for class 'ProcessGraphParameter'
acosh(x)

## S3 method for class 'ProcessNode'
asinh(x)

## S3 method for class 'ProcessGraphParameter'
asinh(x)

## S3 method for class 'ProcessNode'
atanh(x)

## S3 method for class 'ProcessGraphParameter'
atanh(x)
```

```
## S3 method for class 'ProcessNode'
cumsum(x)

## S3 method for class 'ProcessGraphParameter'
cumsum(x)

## S3 method for class 'ProcessNode'
cummin(x)

## S3 method for class 'ProcessGraphParameter'
cummin(x)

## S3 method for class 'ProcessNode'
cummax(x)

## S3 method for class 'ProcessGraphParameter'
cummax(x)

## S3 method for class 'ProcessNode'
cumprod(x)

## S3 method for class 'ProcessGraphParameter'
cumprod(x)

## S3 method for class 'ProcessGraphParameter'
x[i, ..., drop = TRUE]

## S3 method for class 'ProcessNode'
!x

## S3 method for class 'ProcessGraphParameter'
!x

## S3 method for class 'ProcessNode'
quantile(x, ...)

## S3 method for class 'ProcessGraphParameter'
quantile(x, ...)
```

Arguments

x	ProcessGraphParameter, ProcessNode or a list or vector. Passes internal data to the function
...	further arguments to pass on, see the documentation of primitive functions of R for further information
digits	the amount of decimal digits to round to
base	the base of the exponential operation
i	the index of the element in a vector or list

drop	listed for completeness but not used in openEO processes.

Value

a ProcessNode

---

update_job *Modifies a job with given parameter*

Description

The function modifies a stores a job with a given parameter. The dot parameter contains all the values that will be replaced or removed. The return shows a message of result or failure.

Usage

```
update_job(
  id,
  title = NULL,
  description = NULL,
  process = NULL,
  plan = NULL,
  budget = NULL,
  con = NULL,
  ...
)
```

Arguments

id	the job id of a created job
title	update title for the job
description	update description
process	A Graph(), a function returning a ProcessNode() as an endpoint, the ProcessNode() will return the results or a self defined Process()
plan	replaces plan with the set value
budget	replaces or sets the credits that can be spent at maximum
con	connected and authenticated openEO client (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

Details

The '...' operator shall contain all the values that are to be replaced in the job. There are some reserved keys. The 'process_graph' option will replace the process graph with a newly defined one, therefore the process graph needs to be a Graph object. The 'format' option will change the desired output format. All other parameter will be assumed to be special output parameter. Remember, you don't need to specify a process graph or graph_id, e.g. if you just want to update the output format. To leave parameter unchanged, then don't mention it. If you want to delete some, then set them to NA.

---

update_service *Modifies a service*

Description

The function updates a service with the given information. If a parameter is NULL then it will not be overwritten on the back-end. If the parameter is set to NA then the value on the back-end will be deleted and set to NULL.

Usage

```
update_service(
  service,
  type = NULL,
  graph = NULL,
  title = NULL,
  description = NULL,
  enabled = NULL,
  configuration = NULL,
  plan = NULL,
  budget = NULL,
  con = NULL,
  ...
)
```

Arguments

service	the Service or its ID
type	character - the OGC web service type name to be created
graph	A Graph(), a function returning a ProcessNode() as an endpoint or the ProcessNode() will return the results
title	character (optional) - the title of for the service
description	character (optional) - the description for the service
enabled	logical - whether the service shall be active for querying or disabled
configuration	a list of service creation configuration

plan	character - the billing plan
budget	numeric - the amount of credits that can be spent for this service
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

Value

Service object

`update_user_process` *Update an user defined process*

Description

You can change details on an already created user defined process. You can either edit the meta data like the summary or the description. Or you can replace the process graph. However, you cannot delete the process graph, but by passing NA to the meta data fields you can empty those fields in the user defined process.

Usage

```
update_user_process(
  id,
  graph = NULL,
  summary = NULL,
  description = NULL,
  con = NULL,
  ...
)
```

Arguments

id	process graph id
graph	a process graph definition created by combining 'process()', 'collection()' or using a ProcessGraphBuilder
summary	summary of the process graph (optional)
description	description of the process graph (optional)
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

upload_file *Upload data into the users workspace*

Description

This function sends the file retrieved by the 'content' parameter to the specified target location (relative file path in the user workspace) on the back-end.

Usage

```
upload_file(
  content,
  target,
  encode = "raw",
  mime = "application/octet-stream",
  con = NULL
)
```

Arguments

content	the file path of the file to be uploaded
target	the relative server path location for the file, e.g. where to find the file in the users workspace
encode	the encoding type used to upload the data, e.g. 'multipart','form','json','raw' ('raw' by default)
mime	mime type used in upload_file ('application/octet-stream' as a default)
con	authorized Connection (optional) otherwise active_connection() is used.

Value

the relative file path on the server

UserProcessCollection *User Defined Process Collection*

Description

This object contains template functions from the users stored user defined processes (UDP), which can be reused in other process graphs.

Details

This object is an unlocked R6 object, that allows us to add new functions to this object at runtime. It is structured in the same way as the `ProcessCollection()` for predefined processes by the openEO back-end. A `UserProcessCollection()` is usually created at `user_processes()`. If you have submitted new user defined processes to the back-end, make sure to call `user_processes()` again to fetch the latest status.

Methods

`$new(con = NULL)` The object creator created an openEO connection.

Arguments

con optional - an active and authenticated Connection (optional) otherwise `active_connection()` is used.

`user_processes` *Process collection for user defined processes*

Description

The created process graphs via `create_user_process()` at the openEO service are user defined processes. They can be used for the creation of process graphs themselves. For processes provided by the particular openEO service the `processes()` function can be used to obtain a builder for those processes. Analogous to this idea, this function creates a builder object for user defined processes listed and described in `describe_user_process()` and `list_user_processes()`.

Usage

```
user_processes(con = NULL)
```

Arguments

con	a connection to an openEO back-end (optional). Otherwise `active_connection()` is used in order to access personal user defined processes. You need to be logged in

Value

`UserProcessCollection()`

validate_process *Validate a user process*

Description

Sends the process graph as a user process to the openEO service and validates it with the predefined and user-defined processes of the service.

Usage

```
validate_process(graph, con = NULL, ...)
```

Arguments

graph	the process graph that will be sent to the service to be validated
con	connected and authorized openEO client object (optional) otherwise active_connection() is used.
...	additional parameters passed to jsonlite::toJSON() (like 'digits')

variables *Lists the defined variables for a graph*

Description

The function creates a list of the defined (but not necessarily used) variables of a process graph.

Usage

```
variables(x)
```

Arguments

x	a process graph object or a process node

Value

a named list of Variables

VectorCube *VectorCube*

Description

Inheriting from `Argument()` in order to represent a vector cube. This is analogous to the `RasterCube()`.

Value

Object of `R6Class()` representing a vector cube.

See Also

`Array()`, `Integer()`, `EPSGCode()`, `String()`, `Number()`, `Date()`, `RasterCube()`, `VectorCube()`, `ProcessGraphArgument()`, `ProcessGraphParameter()`, `OutputFormatOptions()`, `GeoJson()`, `Boolean()`, `DateTime()`, `Time()`, `BoundingBox()`, `Kernel()`, `TemporalInterval()`, `TemporalIntervals()`, `CollectionId()`, `OutputFormat()`, `AnyOf()`, `ProjDefinition()`, `UdfCodeArgument()`, `UdfRuntimeArgument()` and `UdfRuntimeVersionArgument()`,`TemporalIntervals()`, `MetadataFilter()`

Index