Package 'sevenbridges2'

July 23, 2025

Type Package

Title The 'Seven Bridges Platform' API Client

Version 0.4.0

Maintainer Marko Trifunovic <marko.trifunovic@velsera.com>

Description R client and utilities for 'Seven Bridges Platform' API, from 'Cancer Genomics Cloud' to other 'Seven Bridges' supported platforms. API documentation is hosted publicly at <https://docs.sevenbridges.com/docs/the-api>.

License Apache License 2.0

Encoding UTF-8

VignetteBuilder knitr

URL https://www.sevenbridges.com, https://sbg.github.io/sevenbridges2/, https://github.com/sbg/sevenbridges2

Depends R (>= 4.2.0)

RoxygenNote 7.3.2

Imports httr, R6, purrr, jsonlite, cli, rlang, curl, glue, stringr, utils, checkmate, DescTools, yaml, readr, data.table

Suggests knitr, rmarkdown, testthat (>= 3.0.0), stringi, withr, remotes, pkgdown

Config/testthat/edition 3

Config/testthat/parallel true

Language en-US

BugReports https://github.com/sbg/sevenbridges2/issues

NeedsCompilation no

Author Marko Trifunovic [aut, cre], Marija Gacic [aut], Vladimir Obucina [aut], Velsera [cph, fnd]

Repository CRAN

Date/Publication 2025-03-25 22:50:02 UTC

Contents

api *Core HTTP logic for Seven Bridges API*

Description

Used for advanced users and the core method for higher level API in this package.

Usage

```
api(
  token = NULL,
  path = NULL,
  method = c("GET", "POST", "PUT", "DELETE", "PATCH"),
  query = NULL,
  body = list(),
  encode = c("json", "form", "multipart"),
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  advance_access = getOption("sevenbridges2")$advance_access,
  authorization = FALSE,
  fields = "_all",
  base_url = NULL,
  url = NULL,
  ...
)
```

Arguments

token	API authentication token or access_token for Seven Bridges single sign-on. Authentication token uniquely identifies you on the Seven Bridges Platform and has all your data access, app management and task execution permissions. Read more about its usage here.
path	Path connected with base_url.
method	One of "GET", "POST", "PUT", "DELETE", or "PATCH".
query	Query parameters passed to httr package GET/POST call.
body	Body content passed to httr package GET/POST/PUT/DELETE/PATCH call.
encode	If the body is a named list, how should it be encoded? Can be one of "json" (application/json), "form" (application/x-www-form-urlencoded), or "multipart" (multipart/form-data). Default is "json". For "multipart", list elements can be strings or objects created by httr::upload_file(). For "form", elements are coerced to strings and escaped, use I() to prevent double-escaping. For "json", parameters are automatically "unboxed" (i.e. length 1 vectors are converted to scalars). To preserve a length 1 vector as a vector, wrap in I().
limit	The maximum number of collection items to return for a single request. Minimum value is 1. The maximum value is 100 and the default value is 50. This is a pagination-specific attribute.

offset	The zero-based starting index in the entire collection of the first item to return. The default value is 0. This is a pagination-specific attribute.
advance_access	Enable advance access features? Default is FALSE.
authorization	Is the token an API authentication token (FALSE) or an access token from the Seven Bridges single sign-on (TRUE)?
fields	Selector specifying a subset of fields to include in the response. All API calls take this optional query parameter. This parameter enables you to specify the fields you want to be returned when listing resources (e.g. all your projects) or getting details of a specific resource (e.g. a given project). For example, fields="id,name,size" to return the fields id, name and size for files. Default value is set to _all, so all fields are always returned for each resource. More details please check here.
base_url	Platform URL, default is NULL.
url	Full url of the resource. If url is provided, other parameters like base_url, path, query, limit, offset and fields will be ignored.
...	Other arguments passed to GET/POST/PUT/DELETE/PATCH call.

Value

Response in form of a list.

References

https://docs.sevenbridges.com/page/api

Examples

```
token <- "your_token"
# list projects
## Not run:
api(token = token, path = "projects", method = "GET")

## End(Not run)
```

App *R6 Class representing an app*

Description

R6 Class representing a resource for managing apps.

Super class

sevenbridges2::Item -> App

Public fields

`URL` List of URL endpoints for this resource.

`id` Character used as an app ID - short app name.

`project` Project ID if any, when returned by an API call.

`name` App name.

`revision` App's revision number.

`copy_of` The original application of which this is a copy.

`latest_revision` App's latest revision number.

`raw` App's raw CWL (JSON or YAML).

Methods

Public methods:

- `App$new()`
- `App$print()`
- `App$reload()`
- `App$copy()`
- `App$get_revision()`
- `App$create_revision()`
- `App$sync()`
- `App$input_matrix()`
- `App$output_matrix()`
- `App$create_task()`
- `App$clone()`

Method `new()`**:** Create a new App object.

Usage:

```
App$new(res = NA, ...)
```

Arguments:

`res` Response containing App object information.

`...` Other response arguments.

Returns: A new `App` object.

Method `print()`**:** Print method for App class.

Usage:

```
App$print()
```

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
```

```
  auth = auth,
  response = attr(x, "response")
 )
 app_object$print()
}
```

Method `reload()`**:** Reload App object information. Suitable also for loading raw CWL in the 'raw' field, if it's not already populated.

Usage:

```
App$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `App` object.

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
  res = x,
  href = x$href,
  auth = auth,
  response = attr(x, "response")
 )
 app_object$reload()
}
```

Method `copy()`**:** A method that copies the current app to the specified project.

Usage:

```
App$copy(project, name = NULL, strategy = "clone", use_revision = FALSE, ...)
```

Arguments:

`project` Project object or project ID. If you opt for the latter, remember that the project ID should be specified in `<project_owner>/<project-name>` format, e.g. `rfranklin/my-project`, or as `<division>/<project-name>` depending on the account type.

`name` The new name for the app in the target project (optional).

`strategy` The method for copying the app. Supported strategies:

- `clone` - copy all revisions; get updates from the same app as the copied app (default)
- `direct`: copy latest revision; get updates from the copied app
- `clone_direct`: copy all revisions; get updates from the copied app
- `transient`: copy latest revision; get updates from the same app as the copied app.

`use_revision` Parameter specifying which app's revision should be copied. If set to `FALSE` (default), the latest revision of the app will be copied.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: Copied `App` object.

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 app_object$copy(project)
}
```

Method `get_revision()`**:** Get app's revision.

Usage:

```
App$get_revision(revision = self$revision, in_place = FALSE, ...)
```

Arguments:

`revision` Revision of the app.

`in_place` If `TRUE`, replace current app object with new for specified app revision.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Details: This call allows you to obtain a particular revision of an app, which is not necessarily the most recent version.

Returns: App object.

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 app_object$get_revision()
}
```

Method `create_revision()`**:** Create a new app revision.

Usage:

```
App$create_revision(
  raw = NULL,
  from_path = NULL,
  raw_format = c("JSON", "YAML"),
  in_place = FALSE,
  ...
)
```

Arguments:

`raw` A list containing a raw CWL for the app revision you are about to create. To generate such a list, you might want to load some existing JSON / YAML file. In case that your CWL file is in JSON format, please use the `fromJSON` function from the `jsonlite` package to minimize potential problems with parsing the JSON file. If you want to load a CWL file in YAML format, it is highly recommended to use the `read_yaml` function from the `yaml` package. Keep in mind that this parameter should not be used together with the `file_path` parameter.

`from_path` A path to a file containing the raw CWL for the app (JSON or YAML). This parameter should not be used together with the `raw` parameter.

`raw_format` The type of format used (JSON or YAML).

`in_place` If `TRUE`, replace current app object with newly created revision.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Details: This call creates a new revision for an existing app. It adds a new CWL app description, and stores it as the named revision for the specified app. The revision number must not already exist and should follow the sequence of previously created revisions.

More documentation about how to create the app via API can be found here.

Returns: App object.

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 # Create App object using raw CWL
 app_object$create_revision(raw)
}
```

Method `sync()`**:** Synchronize a copied app with its parent app.

Usage:

```
App$sync(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Details: This call synchronizes a copied app with the source app from which it has been copied.

Returns: App object.

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
```

```
    href = x$href,
    auth = auth,
    response = attr(x, "response")
  )

  app_object$sync()
}
```

Method `input_matrix()`**:** Get an input matrix for the app, listing expected inputs (required or optional) along with their types, descriptions, etc.

Usage:

```
App$input_matrix()
```

Returns: Data frame.

Method `output_matrix()`**:** Get an output matrix for the app, listing expected outputs of tasks that run this app, along with their types, descriptions, etc.

Usage:

```
App$output_matrix()
```

Returns: Data frame.

Method `create_task()`**:** This call creates a new task. You can create either a single task or a batch task by using the app's default batching, override batching, or disable batching completely. A parent task is a task that specifies criteria by which to batch its inputs into a series of further sub-tasks, called child tasks. The documentation on batching tasks for more details on batching criteria.

Usage:

```
App$create_task(
  project,
  revision = NULL,
  name = NULL,
  description = NULL,
  execution_settings = NULL,
  inputs = NULL,
  output_location = NULL,
  batch = NULL,
  batch_input = NULL,
  batch_by = NULL,
  use_interruptible_instances = NULL,
  action = NULL,
  ...
)
```

Arguments:

`project` The ID string of a project or a Project object where you want to create the task in.

`revision` The app revision (version) number.

`name` The name of the task.

`description` An optional description of the task.

`execution_settings` Named list with detailed task execution parameters. Detailed task execution parameters:

- `instance_type`: Possible value is the specific instance type, e.g. `"instance_type" = "c4.2xlarge;ebs-gp2;2000"`;
- `max_parallel_instances`: Maximum number of instances running at the same time. Takes any integer value equal to or greater than 1, e.g. `"max_parallel_instances" = 2.`;
- `use_memoization`: Set to `FALSE` by default. Set to `TRUE` to enable memoization;
- `use_elastic_disk`: Set to `TRUE` to enable Elastic Disk.

Here is an example:

```
execution_settings <- list(
  "instance_type" = "c4.2xlarge;ebs-gp2;2000",
  "max_parallel_instances" = 2,
  "use_memoization" = TRUE,
  "use_elastic_disk" = TRUE
  )
```

`inputs` List of objects. See the section on specifying task inputs for information on creating task input objects. Here is an example with various input types:

```
inputs <- list(
  "input_file"= "<file_id/file_object>",
  "input_directory" = "<folder_id/folder_object>",
  "input_array_string" = list("<string_elem_1>", "<string_elem_2>"),
  "input_boolean" = TRUE,
  "input_double" = 54.6,
  "input_enum" = "enum_1",
  "input_float" = 11.2,
  "input_integer" = "asdf",
  "input_long" = 4212,
  "input_string" = "test_string",
  "input_record" = list(
    "input_record_field_file" = "<file_id/file_object>",
    "input_record_field_integer" = 42
   )
  )
```

`output_location` The output location list allows you to define the exact location where your task outputs will be stored. The location can either be defined for the entire project using the main_location parameter, or individually per each output node, by setting the nodes_override parameter to true and defining individual output node locations within nodes_location. See below for more details.

- `main_location` - Defines the output location for all output nodes in the task. Can be a string path within the project in which the task is created, for example `/Analysis/<task_id>_<task_name>/` or a path on an attached volume, such as `volumes://volume_name/<project_id>/html`. Parts of the path enclosed in angle brackets <> are tokens that are dynamically replaced with corresponding values during task execution.

- main_location_alias: The string location (path) in the project that will point to the actual location where the outputs are stored. Used if main_location is defined as a volume path (starting with volumes://), to provide an easy way of accessing output data directly from project files.
- nodes_override: Enables defining of output locations for output nodes individually through nodes_location (see below). Set to TRUE to be able to define individual locations per output node. Default: FALSE. Even if nodes_override is set to TRUE, it is not necessary to define output locations for each of the output nodes individually. Data from those output nodes that don't have their locations explicitly defined through nodes_location is either placed in main_location (if defined) or at the project files root if a main output location is not defined for the task.
- nodes_location: List of output paths for individual task output nodes in the following format for each output node:

```
<output-node-id> = list(
"output_location" = "<output-path>",
"output_location_alias" = "<alias-path>"
)
```

Example:

```
b64html = list(
 "output_location" = "volumes://outputs/tasks/mar-19",
 "output_location_alias" = "/rfranklin/tasks/picard"
)
```

In the example above, b64html is the ID of the output node for which you want to define the output location, while the parameters are defined as follows:

  - output_location - Can be a path within the project in which the task is created, for example /Analysis/<task_id>_<task_name>/ or a path on an attached volume, such as volumes://volume_name/<project_id>/html. Also accepts tokens.
  - output_location_alias - The location (path) in the project that will point to the exact location where the output is stored. Used if output_location is defined as a volume path (starting with volumes://).

batch This is set to FALSE by default. Set to TRUE to create a batch task and specify the batch_input and batch-by criteria as described below.

batch_input The ID of the input on which you wish to batch. You would typically batch on the input consisting of a list of files. If this parameter is omitted, the default batching criteria defined for the app will be used.

batch_by Batching criteria in form of list. For example:

```
batch_by = list(
  type = "CRITERIA",
  criteria = list("metadata.condition")
)
```

use_interruptible_instances This field can be TRUE or FALSE. Set this field to TRUE to allow the use of spot instances.

action If set to run, the task will be run immediately upon creation.

... Other arguments that can be passed to core api() function like 'fields', etc.

Returns: Task object.

Examples:

```
\dontrun{
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 # Create a DRAFT task
 app_object$create_task(project = project)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
App$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `App$print`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 app_object$print()

## End(Not run)

## ------------------------------------------------
## Method `App$reload`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
```

```
 )
 app_object$reload()

## End(Not run)

## ------------------------------------------------
## Method `App$copy`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 app_object$copy(project)

## End(Not run)


## ------------------------------------------------
## Method `App$get_revision`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 app_object$get_revision()

## End(Not run)


## ------------------------------------------------
## Method `App$create_revision`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 # Create App object using raw CWL
 app_object$create_revision(raw)
```

```
## End(Not run)


## ------------------------------------------------
## Method `App$sync`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )

  app_object$sync()

## End(Not run)


## ------------------------------------------------
## Method `App$create_task`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 app_object <- App$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 # Create a DRAFT task
 app_object$create_task(project = project)

## End(Not run)
```

Apps *R6 Class representing the apps endpoint*

Description

R6 Class representing the apps resource endpoint.

Super class

sevenbridges2::Resource -> Apps

Public fields

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Apps$new()`
- `Apps$query()`
- `Apps$get()`
- `Apps$copy()`
- `Apps$create()`
- `Apps$clone()`

Method `new()`**:** Create a new Apps resource object.

Usage:

```
Apps$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** This call lists all the apps available to you.

Usage:

```
Apps$query(
  project = NULL,
  visibility = c("private", "public"),
  query_terms = NULL,
  id = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  fields = "!raw",
  ...
)
```

Arguments:

`project` Project ID string in the form `<project_owner>/<project_short_name>` or `<division_name>/<project_short_name>` or Project object, to restrict the results to apps from that project only.

`visibility` Set this to `public` to see all public apps on the Seven Bridges Platform.

`query_terms` A list of search terms used to filter apps based on their details. Each term is case-insensitive and can relate to the app's name, label, toolkit, toolkit version, category, tagline, or description. You can provide a single term (e.g., `list("Compressor")`) or multiple terms (e.g., `list("Expression", "Abundance")`) to search for apps that match all the specified terms. If a term matches any part of the app's details, the app will be included in the results. Search terms can also include phrases (e.g., `list("Abundance estimates input")`), which will search for exact matches within app descriptions or other fields.

`id` Use this parameter to query apps based on their ID.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`fields` Selector specifying a subset of fields to include in the response. For querying apps, it is set to return all fields except 'raw' which stores CWL as a list. Be cautious when requesting all fields, as this API request may take a long time to execute.

`...` Other arguments that can be passed to core `api()` function.

Returns: `Collection` containing `App` objects.

Examples:

```
\dontrun{
 apps_object <- Apps$new(
   auth = auth
 )

 # List public apps
 apps_object$query(visibility = "public")
}
```

Method `get()`**:** This call returns information about the specified app. The app must be in a project you can access. It could be an app uploaded to the Seven Bridges Platform by a project member or a public app copied into the project.
You can find more details about this operation in our API documentation.

Usage:

```
Apps$get(id, revision = NULL, ...)
```

Arguments:

`id` The full `<project_id>/<app_short_name>` path for this API call is known as App ID. You can also get the App ID for an app by making the call to list all apps available to you.

`revision` The number of the app revision you want to get.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `App` object.

Examples:

```
\dontrun{
 apps_object <- Apps$new(
   auth = auth
 )

 # Get app object
 apps_object$get(id = "<some_id>")
}
```

Method `copy()`**:** This call copies the specified app to the specified project. The app must be in a project you can access. It could be an app uploaded to the Seven Bridges Platform by a project member or a public app copied into the project.

Usage:

```
Apps$copy(
  app,
  project,
  name = NULL,
  strategy = c("clone", "direct", "clone_direct", "transient"),
  ...
)
```

Arguments:

`app` App object or the short name of the app you are copying. Optionally, to copy a specific revision of the app, use the `<app_short_name>/<revision_number>` format, for example `rfranklin/my-project/bamtools-index-2-4-0/1`

`project` The Project object or project ID you want to copy the app to.

`name` The new name the app will have in the target project. If its name will not change, omit this key.

`strategy` The method for copying the app. Can be one of:

- `clone` : copy all revisions; get updates from the same app as the copied app (default);
- `direct`: copy latest revision; get updates from the copied app;
- `clone_direct`: copy all revisions; get updates from the copied app;
- `transient`: copy latest revision; get updates from the same app as the copied app.

Read more about the strategies here.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: Copied App object.

Examples:

```
\dontrun{
 apps_object <- Apps$new(
   auth = auth
 )
 # Copy app object to a project
 apps_object$copy(app = app, project = project)
}
```

Method `create()`**:** This call allows you to add an app using raw CWL.

Usage:

```
Apps$create(
  raw = NULL,
  from_path = NULL,
  project,
  name,
  raw_format = c("JSON", "YAML"),
  ...
)
```

Arguments:

`raw` The body of the request should be a CWL app description saved as a `JSON` or `YAML` file. For a template of this description, try making the call to get raw CWL for an app about an app already in one of your projects. Shouldn't be used together with `from_path` parameter.

`from_path` File containing CWL app description. Shouldn't be used together with raw parameter.

`project` String project ID or Project object in which you want to store the app.

`name` A short name for the app (without any non-alphanumeric characters or spaces)

`raw_format` The type of format used (`JSON` or `YAML`).

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `App` object.

Examples:

```
\dontrun{
 apps_object <- Apps$new(
   auth = auth
 )

 # Create new app object
 apps_object$create(
  raw = raw,
  project = project,
  name = name,
  raw_format = "YAML"
 )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Apps$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Apps$query`
## ------------------------------------------------

## Not run:
 apps_object <- Apps$new(
   auth = auth
 )

 # List public apps
 apps_object$query(visibility = "public")

## End(Not run)
```

```
## ------------------------------------------------
## Method `Apps$get`
## ------------------------------------------------

## Not run:
 apps_object <- Apps$new(
   auth = auth
 )

 # Get app object
 apps_object$get(id = "<some_id>")

## End(Not run)


## ------------------------------------------------
## Method `Apps$copy`
## ------------------------------------------------

## Not run:
 apps_object <- Apps$new(
   auth = auth
 )
 # Copy app object to a project
 apps_object$copy(app = app, project = project)

## End(Not run)


## ------------------------------------------------
## Method `Apps$create`
## ------------------------------------------------

## Not run:
 apps_object <- Apps$new(
   auth = auth
 )

 # Create new app object
 apps_object$create(
  raw = raw,
  project = project,
  name = name,
  raw_format = "YAML"
 )

## End(Not run)
```

AsyncJob *R6 Class representing an AsyncJob*

Description

R6 Class representing a resource for managing asynchronous jobs.

Super class

sevenbridges2::Item -> AsyncJob

Public fields

`id` Asynchronous job ID.

`type` The type of job. Can be one of: COPY, DELETE, MOVE.

`state` The following states are available: SUBMITTED, RESOLVING, RUNNING and FINISHED.

`result` The result of the job.

`total_files` The total number of files that were processed for the job.

`completed_files` The number of files that were successfully completed.

`failed_files` The number of files that failed.

`started_on` The time and date the job started.

`finished_on` The time and date the job finished.

Methods

Public methods:

- `AsyncJob$new()`
- `AsyncJob$print()`
- `AsyncJob$reload()`
- `AsyncJob$clone()`

Method `new()`**:** Create a new AsyncJob object.

Usage:

```
AsyncJob$new(res = NA, ...)
```

Arguments:

`res` Response containing AsyncJob object information.

`...` Other response arguments.

Returns: A new `AsyncJob` object.

Method `print()`**:** Print method for AsyncJob class.

Usage:

```
AsyncJob$print()
```

Examples:

```
\dontrun{
 # x is API response when app is requested
 asyncjob_object <- AsyncJob$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 asyncjob_object$print()
}
```

Method `reload()`**:** Reloads AsyncJob object information.

Usage:

```
AsyncJob$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `AsyncJob` object.

Examples:

```
\dontrun{
 # x is API response when AsyncJob is requested
 asyncjob_object <- AsyncJob$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 asyncjob_object$reload()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AsyncJob$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AsyncJob$print`
## ------------------------------------------------

## Not run:
 # x is API response when app is requested
 asyncjob_object <- AsyncJob$new(
```

```
  res = x,
  href = x$href,
  auth = auth,
  response = attr(x, "response")
 )
 asyncjob_object$print()

## End(Not run)

## ------------------------------------------------
## Method `AsyncJob$reload`
## ------------------------------------------------

## Not run:
 # x is API response when AsyncJob is requested
 asyncjob_object <- AsyncJob$new(
  res = x,
  href = x$href,
  auth = auth,
  response = attr(x, "response")
 )
 asyncjob_object$reload()

## End(Not run)
```

`Auth` *R6 Class Representing Authentication Object*

Description

Authentication object with methods for accessing API endpoints. Every object could be requested from this Auth object and any action could start from this object using cascading style. Please check `vignette("Authentication_and_Billing", package = "sevenbridges2")` for more information.

Details

This is the main object for authentication to platforms powered by Seven Bridges.

Public fields

`from` Authentication method.

`platform` The platform to use.

`url` Base URL for API.

`sysenv_url` Name of the system environment variable storing the API base URL.

`sysenv_token` Name of the system environment variable storing the auth token.

`config_file` Location of the user configuration file.

`profile_name` Profile name in the user configuration file.

`fs` FS (FileSystem) object, for mounting and unmounting the file system.

`authorization` Is the `token` an API authentication token (`FALSE`) or an access token from the Seven Bridges single sign-on (`TRUE`)?

`projects` Projects object, for accessing projects resources on the platform.

`files` Files object, for accessing files resources on the platform.

`apps` Apps object, for accessing apps resources on the platform.

`volumes` Volumes object, for accessing volumes resources on the platform.

`tasks` Tasks object, for accessing tasks resources on the platform.

`imports` Storage imports object, for accessing volume imports resources on the platform.

`exports` Storage exports object, for accessing volume exports resources on the platform.

`invoices` Invoices object, for accessing invoice resources on the platform.

`billing_groups` Billing_groups object, for accessing billing groups resources on the platform.

`divisions` Divisions object, for accessing divisions resources on the platform.

`teams` Teams object, for accessing teams resources on the platform.

Methods

Public methods:

- `Auth$new()`
- `Auth$get_token()`
- `Auth$api()`
- `Auth$user()`
- `Auth$rate_limit()`
- `Auth$upload()`
- `Auth$list_ongoing_uploads()`
- `Auth$upload_abort()`
- `Auth$send_feedback()`
- `Auth$clone()`

Method `new()`**:** Create a new Seven Bridges API Authentication object. All methods can be accessed through this object.

Usage:

```
Auth$new(
  from = c("direct", "env", "file"),
  platform = NA,
  url = NA,
  token = NA,
  sysenv_url = NA,
  sysenv_token = NA,
  config_file = NA,
  profile_name = NA,
  fs = NA,
  authorization = FALSE
)
```

Arguments:

`from` Authentication method. Could be:

- `"direct"` - pass the credential information to the arguments directly,
- `"env"` - read from pre-set system environment variables, or
- `"file"` - read configurations from a credentials file.

Default is `"direct"`.

`platform` The platform to use. If neither `platform` nor `url` is specified the default is `"aws-us"` (Seven Bridges Platform - US). Other possible values include:

- `"aws-eu"` - Seven Bridges Platform - EU,
- `"cgc"` - Cancer Genomics Cloud,
- `"ali-cn"` - Seven Bridges Platform - China,
- `"cavatica"` - Cavatica, and
- `"f4c"` - BioData Catalyst Powered by Seven Bridges.

`url` Base URL for API. Please only use this when you want to specify a platform that is not in the `platform` list above, while leaving `platform` unspecified.

`token` API authentication token or `access_token` for Seven Bridges single sign-on. Authentication token uniquely identifies you on the Seven Bridges Platform and has all your data access, app management and task execution permissions. Read more about its usage here.

`sysenv_url` Name of the system environment variable storing the API base URL. By default: `"SB_API_ENDPOINT"`.

`sysenv_token` Name of the system environment variable storing the auth token. By default: `"SB_AUTH_TOKEN"`.

`config_file` Location of the user configuration file.
By default: `"~/.sevenbridges/credentials"`.

`profile_name` Profile name in the user configuration file. The default value is `"default"`.

`fs` FS (FileSystem) object, for mount and unmount file system.

`authorization` Is the `token` an API authentication token (`FALSE`) or an access token from the Seven Bridges single sign-on (`TRUE`)?

Returns: `Auth` class object.

Examples:

```
\dontrun{
 # Multiple ways to create Auth object

 # Using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Authenticate using environment variables
 a <- Auth$new(from = "env")

 # Authenticate using file configuration
 a <- Auth$new(from = "file")

}
```

Method `get_token()`**:** Returns the authentication token read from system environment variable.

Usage:

```
Auth$get_token()
```

Returns: An API authentication token in form of a string.

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Get that same token
 a$get_token()
}
```

Method `api()`**:** This method returns all API paths and pass arguments to core `api()` function.

Usage:

```
Auth$api(
  ...,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  fields = "_all"
)
```

Arguments:

`...` Other arguments passed to core `api()` function, like `path`, `query` parameters or full `url` to some resource.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`fields` Selector specifying a subset of fields to include in the response. This parameter enables you to specify the fields you want to be returned when listing resources (e.g. all your projects) or getting details of a specific resource (e.g. a given project).

For example, `fields="id,name,size"` to return the fields id, name and size for files. Default value is set to `_all`, so all fields are always returned for each resource. More details please check general API documentation.

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
```

```
  platform = "aws-us"
 )

 # Create API request using request parameters directly
 a$api(params)
}
```

Method `user()`**:** Get details about the authenticated user.

Usage:

```
Auth$user(username = NULL)
```

Arguments:

`username` The username of a user for whom you want to get basic account information. If not provided, information about the currently authenticated user will be returned.

Returns: `User` class object.

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Get information about the currently authenticated user
 a$user()
}
```

Method `rate_limit()`**:** Get information about current rate limit.

This call returns information about your current rate limit. This is the number of API calls you can make in one hour. This call also returns information about your current instance limit.

Usage:

```
Auth$rate_limit()
```

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Get current rate limit
 a$rate_limit()
}
```

Method `upload()`**:** This method allows you to upload a single file from your local computer to the Platform.

Usage:

```
Auth$upload(
  path,
  project = NULL,
  parent = NULL,
  filename = NULL,
  overwrite = FALSE,
  part_size = getOption("sevenbridges2")$RECOMMENDED_PART_SIZE,
  init = FALSE
)
```

Arguments:

`path` File path on local disk.

`project` `Project` object or its ID. Project should not be used together with parent. If parent is used, the call will upload the file to the specified Platform folder, within the project to which the folder belongs. If project is used, the call will upload the file to the root of the project's files.

`parent` Parent folder object (of `File` class) or its ID. Should not be used together with project. If parent is used, the call will upload the file to the specified Platform folder, within the project to which the folder belongs. If project is used, the call will upload the file to the root of the project's files.

`filename` Optional new file name. By default the uploaded file will have the same name as the original file provided with the `path` parameter. If its name will not change, omit this key.

`overwrite` In case there is already a file with the same name in the selected platform project or folder, this option allows you to control whether that file will be overwritten or not. If overwrite is set to `TRUE` and a file already exists under the name specified in the request, the existing file will be deleted and a new one created in its place.

`part_size` The preferred size for upload parts in bytes. If omitted or set to a value that is incompatible with the cloud storage provider, a default value will be used.

`init` If `TRUE`, the method will initialize and return the Upload object and stop. If `FALSE`, the method will return the Upload object and start the upload process immediately.

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Create upload job and set destination project
 upload_job <- a$upload(
   path = "/path/to/your/file.txt",
   project = destination_project,
   overwrite = TRUE,
   init = TRUE
```

```
 )
}
```

Method `list_ongoing_uploads()`**:** This method returns the list of all ongoing uploads.

Usage:

```
Auth$list_ongoing_uploads()
```

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # List ongoing uploads
 a$list_ongoing_uploads()
}
```

Method `upload_abort()`**:** This call aborts an ongoing multipart upload.

Usage:

```
Auth$upload_abort(upload_id)
```

Arguments:

`upload_id` Upload object or ID of the upload process that you want to abort.

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Abort upload
 a$abort_upload(upload_id = "<id_of_the_upload_process>")
}
```

Method `send_feedback()`**:** Send feedback to Seven Bridges.

Send feedback on ideas, thoughts, and problems via the sevenbridges2 API package with three available types: `idea`, `thought`, and `problem`. You can send one feedback item per minute.

Usage:

```
Auth$send_feedback(
  text,
  type = c("idea", "thought", "problem"),
  referrer = NULL
)
```

Arguments:

`text` Specifies the content for the feedback i.e. feedback text.

`type` Specifies the type of feedback. The following are available: `idea`, `thought` and `problem`.

`referrer` The name of the person submitting the feedback.

Examples:

```
\dontrun{
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Send feedback
 a$send_feedback(
  "This is a test for sending feedback via API.",
  type = "thought"
 )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Auth$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Auth$new`
## ------------------------------------------------

## Not run:
 # Multiple ways to create Auth object

 # Using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Authenticate using environment variables
 a <- Auth$new(from = "env")

 # Authenticate using file configuration
 a <- Auth$new(from = "file")


## End(Not run)

## ------------------------------------------------
```

```
## Method `Auth$get_token`
## ------------------------------------------------

## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Get that same token
 a$get_token()

## End(Not run)


## ------------------------------------------------
## Method `Auth$api`
## ------------------------------------------------

## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Create API request using request parameters directly
 a$api(params)

## End(Not run)


## ------------------------------------------------
## Method `Auth$user`
## ------------------------------------------------

## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Get information about the currently authenticated user
 a$user()

## End(Not run)

## ------------------------------------------------
## Method `Auth$rate_limit`
## ------------------------------------------------
```

```
## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Get current rate limit
 a$rate_limit()

## End(Not run)


## ------------------------------------------------
## Method `Auth$upload`
## ------------------------------------------------

## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Create upload job and set destination project
 upload_job <- a$upload(
   path = "/path/to/your/file.txt",
   project = destination_project,
   overwrite = TRUE,
   init = TRUE
 )

## End(Not run)

## ------------------------------------------------
## Method `Auth$list_ongoing_uploads`
## ------------------------------------------------

## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # List ongoing uploads
 a$list_ongoing_uploads()

## End(Not run)

## ------------------------------------------------
## Method `Auth$upload_abort`
## ------------------------------------------------
```

```
## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Abort upload
 a$abort_upload(upload_id = "<id_of_the_upload_process>")

## End(Not run)

## ------------------------------------------------
## Method `Auth$send_feedback`
## ------------------------------------------------

## Not run:
 # Authenticate using authentication token
 a <- Auth$new(
  token = "<your_token>",
  platform = "aws-us"
 )

 # Send feedback
 a$send_feedback(
  "This is a test for sending feedback via API.",
  type = "thought"
 )

## End(Not run)
```

`Billing` *R6 Class representing billing information.*

Description

R6 Class representing a central resource for managing billing groups.

Details

This is the main object for Billing

Super class

`sevenbridges2::Item` -> `Billing`

Public fields

`URL` List of URL endpoints for this resource.

`id` Billing group identifier.

`owner` Username of the user that owns the billing group.

`name` Billing group name.

`type` Billing group type.

`pending` Billing group approval status.

`disabled` Indicator of whether the billing group is disabled.

`balance` Billing group balance.

Methods

Public methods:

- `Billing$new()`
- `Billing$print()`
- `Billing$reload()`
- `Billing$analysis_breakdown()`
- `Billing$storage_breakdown()`
- `Billing$egress_breakdown()`
- `Billing$clone()`

Method `new()`**:** Create a new Billing object.

Usage:

```
Billing$new(res = NA, ...)
```

Arguments:

`res` Response containing Billing object information.

`...` Other response arguments.

Method `print()`**:** Prints billing group information as a bullet list.

Usage:

```
Billing$print()
```

Examples:

```
\dontrun{
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print billing group
 billing_object$print()
}
```

Method `reload()`**:** Reload Billing group object.

Usage:

```
Billing$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'limit', 'offset', 'fields', etc.

Returns: `Billing` object.

Examples:

```
\dontrun{
 # x is API response when billing group is requested
 billing_object <- Billing$new(
                     res = x,
                     href = x$href,
                     auth = auth,
                     response = attr(x, "response")
                    )

 # Reload billing group
 billing_object$reload()
}
```

Method `analysis_breakdown()`**:** Method for getting an analysis breakdown for a billing group.

Usage:

```
Billing$analysis_breakdown(
  date_from = NULL,
  date_to = NULL,
  invoice = NULL,
  fields = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`date_from` A string representing the starting date for retrieving transactions analysis in the following format: mm-dd-yyyy.

`date_to` A string representing the ending date for retrieving transactions analysis in the following format: mm-dd-yyyy.

`invoice` A string representing invoice ID or Invoice object to show a breakdown for the specific invoice. If omitted, the current spending breakdown is returned.

`fields` Selector specifying a subset of fields to include in the response.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function.

Examples:

```
\dontrun{
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get analysis breakdown
 billing_object$analysis_breakdown()
}
```

Method `storage_breakdown()`**:** Method for getting a storage breakdown for a billing group.

Usage:

```
Billing$storage_breakdown(
  date_from = NULL,
  date_to = NULL,
  invoice = NULL,
  fields = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`date_from` A string representing the starting date for retrieving storage analysis in the following format: mm-dd-yyyy.

`date_to` A string representing the ending date for retrieving storage analysis in the following format: mm-dd-yyyy.

`invoice` A string representing invoice ID or Invoice object to show a breakdown for the specific invoice. If omitted, the current spending breakdown is returned.

`fields` Selector specifying a subset of fields to include in the response.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function.

Examples:

```
\dontrun{
```

```
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get storage breakdown
 billing_object$storage_breakdown()
}
```

Method `egress_breakdown()`**:** Method for getting a egress breakdown for a billing group.

Usage:

```
Billing$egress_breakdown(
  date_from = NULL,
  date_to = NULL,
  invoice = NULL,
  fields = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`date_from` A string representing the starting date for retrieving egress analysis in the following format: mm-dd-yyyy.

`date_to` A string representing the ending date for retrieving egress analysis in the following format: mm-dd-yyyy.

`invoice` A string representing invoice ID or Invoice object to show a breakdown for the specific invoice. If omitted, the current spending breakdown is returned.

`fields` Selector specifying a subset of fields to include in the response.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function.

Examples:

```
\dontrun{
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
```

```
                    )

 # Get egress breakdown
 billing_object$egress_breakdown()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Billing$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Billing$print`
## ------------------------------------------------

## Not run:
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print billing group
 billing_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Billing$reload`
## ------------------------------------------------

## Not run:
 # x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload billing group
 billing_object$reload()

## End(Not run)
```

```
## ------------------------------------------------
## Method `Billing$analysis_breakdown`
## ------------------------------------------------

## Not run:
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get analysis breakdown
 billing_object$analysis_breakdown()

## End(Not run)


## ------------------------------------------------
## Method `Billing$storage_breakdown`
## ------------------------------------------------

## Not run:
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get storage breakdown
 billing_object$storage_breakdown()

## End(Not run)


## ------------------------------------------------
## Method `Billing$egress_breakdown`
## ------------------------------------------------

## Not run:
# x is API response when billing group is requested
 billing_object <- Billing$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
```

```
# Get egress breakdown
billing_object$egress_breakdown()

## End(Not run)
```

`Billing_groups` *R6 Class representing billing groups endpoints*

Description

R6 Class representing billing groups resource endpoints.

Super class

`sevenbridges2::Resource` -> `Billing_groups`

Public fields

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Billing_groups$new()`
- `Billing_groups$query()`
- `Billing_groups$get()`
- `Billing_groups$clone()`

Method `new()`**:** Create a new Billing_groups object.

Usage:

```
Billing_groups$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** List all your billing groups, including groups that are pending or have been disabled.

Usage:

```
Billing_groups$query(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like query parameters, 'fields', etc.

Returns: `Collection` of `Billing` groups.

Examples:

```
\dontrun{
 billing_groups_object <- Billing_groups$new(
   auth = auth
 )

 # List all your billing groups
 billing_groups_object$query()
}
```

Method `get()`**:** Retrieve a single billing group, specified by its ID. To find the `billing_group`, use the call `Billing_groups$query()` to list all your billing groups. The information returned includes the billing group owner, the total balance, and the status of the billing group (pending or confirmed).

Usage:

```
Billing_groups$get(id, ...)
```

Arguments:

`id` The ID of the billing group you are querying.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Billing` object.

Examples:

```
\dontrun{
 billing_groups_object <- Billing_groups$new(
   auth = auth
 )

 # Get single billing group
 billing_groups_object$get(id = id)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Billing_groups$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Billing_groups$query`
## ------------------------------------------------

## Not run:
 billing_groups_object <- Billing_groups$new(
   auth = auth
 )

 # List all your billing groups
 billing_groups_object$query()

## End(Not run)


## ------------------------------------------------
## Method `Billing_groups$get`
## ------------------------------------------------

## Not run:
 billing_groups_object <- Billing_groups$new(
   auth = auth
 )

 # Get single billing group
 billing_groups_object$get(id = id)

## End(Not run)
```

`Collection` *R6 Class representing a Collection of objects*

Description

R6 Class representing a resource for managing collections. A wrapper for Seven Bridges pageable resources. Among the actual collection items it contains information regarding the total number of entries available on the server and resource API request URL (href).

Public fields

`href` API request URL.

`items` Items returned in API response.

`links` List of links (hrefs) for next and/or previous page resources.

`total` Total number of items available on the server.

`response` Raw API response.

`auth` Seven Bridges Authentication object.

Methods

Public methods:

- Collection$new()
- Collection$print()
- Collection$next_page()
- Collection$prev_page()
- Collection$all()
- Collection$clone()

Method `new()`**:** Create a new Collection object.

Usage:

```
Collection$new(
  href = NA,
  items = NA,
  links = NA,
  total = NA,
  response = NA,
  auth = NA
)
```

Arguments:

`href` API request URL.

`items` Items returned in API response.

`links` List of links (hrefs) for next and/or previous page resources.

`total` Total number of items available on the server.

`response` Raw API response.

`auth` Seven Bridges Authentication object.

Method `print()`**:** Print method for Collection class.

Usage:

```
Collection$print(n = 10)
```

Arguments:

`n` Number of items to print in console.

Examples:

```
\dontrun{
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                     href = x$href,
                                     items = x$items,
                                     links = x$links,
                                     total = x$total,
                                     auth = auth,
                                     response = attr(x, "response")
                                    )
```

```
 # Print collection object
 collection_object$print()
}
```

Method `next_page()`**:** Returns the next page of results.

Usage:

```
Collection$next_page(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'advanced_access', 'fields', etc.

Examples:

```
\dontrun{
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                    href = x$href,
                                    items = x$items,
                                    links = x$links,
                                    total = x$total,
                                    auth = auth,
                                    response = attr(x, "response")
                                  )

 # Get next page of collection results
 collection_object$next_page()
}
```

Method `prev_page()`**:** Returns the previous page of results.

Usage:

```
Collection$prev_page(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'advanced_access', 'fields', etc.

Examples:

```
\dontrun{
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                    href = x$href,
                                    items = x$items,
                                    links = x$links,
                                    total = x$total,
                                    auth = auth,
                                    response = attr(x, "response")
                                  )
```

```
 # Get previous page of collection results
 collection_object$prev_page()
}
```

Method `all()`**:** Fetches all available items by iterating through all pages. Please be aware of the API rate limit for your request.

Usage:

```
Collection$all(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'advanced_access', 'fields', etc.

Examples:

```
\dontrun{
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                    href = x$href,
                                    items = x$items,
                                    links = x$links,
                                    total = x$total,
                                    auth = auth,
                                    response = attr(x, "response")
                                   )

 # Get all results of collection
 collection_object$all()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Collection$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Collection$print`
## ------------------------------------------------

## Not run:
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                  href = x$href,
                                  items = x$items,
                                  links = x$links,
```

```
                                  total = x$total,
                                  auth = auth,
                                  response = attr(x, "response")
                                )

 # Print collection object
 collection_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Collection$next_page`
## ------------------------------------------------

## Not run:
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                  href = x$href,
                                  items = x$items,
                                  links = x$links,
                                  total = x$total,
                                  auth = auth,
                                  response = attr(x, "response")
                                )

 # Get next page of collection results
 collection_object$next_page()

## End(Not run)


## ------------------------------------------------
## Method `Collection$prev_page`
## ------------------------------------------------

## Not run:
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                  href = x$href,
                                  items = x$items,
                                  links = x$links,
                                  total = x$total,
                                  auth = auth,
                                  response = attr(x, "response")
                                )

 # Get previous page of collection results
 collection_object$prev_page()

## End(Not run)
```

```
## ------------------------------------------------
## Method `Collection$all`
## ------------------------------------------------

## Not run:
 # x is API response when collection object is requested
 collection_object <- Collection$new(
                                    href = x$href,
                                    items = x$items,
                                    links = x$links,
                                    total = x$total,
                                    auth = auth,
                                    response = attr(x, "response")
                                  )

 # Get all results of collection
 collection_object$all()

## End(Not run)
```

`Division` *R6 Class representing a Division*

Description

R6 Class representing a central resource for managing divisions.

Super class

`sevenbridges2::Item` -> `Division`

Public fields

`URL` List of URL endpoints for this resource.

`id` The ID of the division.

`name` Division's name.

Methods

Public methods:

- `Division$new()`
- `Division$print()`
- `Division$reload()`
- `Division$list_teams()`
- `Division$list_members()`
- `Division$remove_member()`

- Division$clone()

Method new(): Create a new Division object.

Usage:

```
Division$new(res = NA, ...)
```

Arguments:

res Response containing the Division object information.

... Other response arguments.

Method print(): Print method for Division class.

Usage:

```
Division$print()
```

Examples:

```
\dontrun{
 division_object <- Division$new(
  res = x,
href = x$href,
auth = auth,
response = attr(x, "response")
 )
 division_object$print()
}
```

Method reload(): Reload Division object information.

Usage:

```
Division$reload(...)
```

Arguments:

... Other arguments that can be passed to core api() function like 'fields', etc. @importFrom rlang inform

Returns: Division object.

Examples:

```
\dontrun{
 division_object <- Division$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 division_object$reload()
}
```

Method list_teams(): This call retrieves a list of all teams in a division that you are a member of. Each team's ID and name will be returned.

Usage:

```
Division$list_teams(list_all = FALSE, ...)
```

Arguments:

`list_all` Boolean. Set this field to `TRUE` if you want to list all teams within the division (regardless of whether you are a member of a team or not). Default value is `FALSE`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: A `Collection` of `Team` objects.

Examples:

```
\dontrun{
  # Get details of a specific division
  division_obj <- a$divisions$get(id = "division-id")

  # Retrieve a list of division teams you are a member of
  division_obj$list_teams()

  # Retrieve a list of all teams within the division regardless of
  # whether you are a member of a team or not
  division_obj$list_teams(list_all = TRUE)
}
```

Method `list_members()`**:** This call retrieves a list of all members of a division. In addition, you can list members with a specific role, e.g. all administrators within a division.

Usage:

```
Division$list_members(
  role = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`role` Filter members by role. Supported roles are `ADMIN`, `MEMBER`, and `EXTERNAL_COLLABORATOR`. If `NULL` (default), members of all roles will be retrieved.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like other query parameters or 'fields', etc.

Returns: A `Collection` of `User` objects.

Examples:

```
\dontrun{
  # Get details of a specific division
  division_obj <- a$divisions$get(id = "division-id")
```

```
  # Retrieve a list of all division members
  division_obj$list_members()

  # Or filter members by role. The following roles are supported:
  # "MEMBER", "ADMIN", and "EXTERNAL_COLLABORATOR"
  division_obj$list_members(role = "ADMIN")
}
```

Method `remove_member()`**:** Removes a specified user from a division. This action revokes the user's membership in the division but does not delete their Platform account. Note that only users with the `ADMIN` role in the division can perform this action.

Usage:

```
Division$remove_member(user)
```

Arguments:

`user` The Seven Bridges Platform username of the user to be removed, specified in the format `division-name/username`, or an object of class `User` that contains the username.

Examples:

```
\dontrun{
  # Retrieve details of a specific division
  division_obj <- a$divisions$get(id = "division-id")

  # Remove a member using their username
  division_obj$remove_member(user = "division-name/username")

  # Remove a member using a User object
  members <- division_obj$list_members(role = "MEMBER")
  member_to_remove <- members$items[[1]]
  division_obj$remove_member(user = member_to_remove)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Division$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Division$print`
## ------------------------------------------------

## Not run:
 division_object <- Division$new(
  res = x,
```

```
href = x$href,
auth = auth,
response = attr(x, "response")
 )
 division_object$print()

## End(Not run)

## ------------------------------------------------
## Method `Division$reload`
## ------------------------------------------------

## Not run:
 division_object <- Division$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 division_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `Division$list_teams`
## ------------------------------------------------

## Not run:
  # Get details of a specific division
  division_obj <- a$divisions$get(id = "division-id")

  # Retrieve a list of division teams you are a member of
  division_obj$list_teams()

  # Retrieve a list of all teams within the division regardless of
  # whether you are a member of a team or not
  division_obj$list_teams(list_all = TRUE)

## End(Not run)


## ------------------------------------------------
## Method `Division$list_members`
## ------------------------------------------------

## Not run:
  # Get details of a specific division
  division_obj <- a$divisions$get(id = "division-id")

  # Retrieve a list of all division members
  division_obj$list_members()
```

```
  # Or filter members by role. The following roles are supported:
  # "MEMBER", "ADMIN", and "EXTERNAL_COLLABORATOR"
  division_obj$list_members(role = "ADMIN")

## End(Not run)


## ------------------------------------------------
## Method `Division$remove_member`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specific division
  division_obj <- a$divisions$get(id = "division-id")

  # Remove a member using their username
  division_obj$remove_member(user = "division-name/username")

  # Remove a member using a User object
  members <- division_obj$list_members(role = "MEMBER")
  member_to_remove <- members$items[[1]]
  division_obj$remove_member(user = member_to_remove)

## End(Not run)
```

`Divisions` *R6 Class representing divisions endpoints.*

Description

R6 Class representing Divisions resource.

Super class

`sevenbridges2::Resource` -> `Divisions`

Public fields

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Divisions$new()`
- `Divisions$query()`
- `Divisions$get()`
- `Divisions$clone()`

Method `new()`**:** Create new Divisions resource object.

Usage:

```
Divisions$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** This call retrieves a list of all divisions you are a member of. Each division's ID, name and URL on platform will be returned.

Usage:

```
Divisions$query()
```

Returns: A `Collection` of `Division` objects.

Examples:

```
\dontrun{
  # Retrieve a list of all divisions you are a member of
  a$Divisions$query()
}
```

Method `get()`**:** This call returns the details of a specified division.

Usage:

```
Divisions$get(id, ...)
```

Arguments:

`id` The ID of the division you are querying. The function also accepts a Division object and extracts the ID.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Division` object.

Examples:

```
\dontrun{
  # Retrieve details of a specified division
  a$Divisions$get(id = "division-id")
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Divisions$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Divisions$query`
## ------------------------------------------------

## Not run:
  # Retrieve a list of all divisions you are a member of
```

```
  a$Divisions$query()

## End(Not run)

## ------------------------------------------------
## Method `Divisions$get`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified division
  a$Divisions$get(id = "division-id")

## End(Not run)
```

`Export` *R6 Class representing an Export*

Description

R6 Class representing a resource for managing volume export jobs.

Super class

`sevenbridges2::Item` -> `Export`

Public fields

`URL` List of URL endpoints for this resource.

`id` Export job string identifier.

`state` The state of the export job. Possible values are:

- `PENDING`: the export is queued;
- `RUNNING`: the export is running;
- `COMPLETED`: the export has completed successfully;
- `FAILED`: the export has failed.

`source` List containing the source file ID that is being exported to the volume.

`destination` List containing the destination volume ID and location (file name) on the volume where the file is being exported.

`overwrite` Indicates whether the exported file name was overwritten if another file with the same name already existed on the volume.

`started_on` Time when the export job started.

`finished_on` Time when the export job ended.

`properties` List of volume properties set.

`error` In case of error in the export job, standard API error is returned here.

`result` File object that was exported.

Methods

Public methods:

- `Export$new()`
- `Export$print()`
- `Export$reload()`
- `Export$clone()`

Method `new()`**:** Create a new Export object.

Usage:

```
Export$new(res = NA, ...)
```

Arguments:

`res` Response containing Export job information.

`...` Other response arguments.

Method `print()`**:** Print method for Export class.

Usage:

```
Export$print()
```

Examples:

```
\dontrun{
# x is API response when export is requested
export_object <- Export$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print export object
 export_object$print()
}
```

Method `reload()`**:** Refresh the Export object with updated information.

Usage:

```
Export$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Export` object.

Examples:

```
\dontrun{
# x is API response when export is requested
export_object <- Export$new(
                    res = x,
```

```
                      href = x$href,
                      auth = auth,
                      response = attr(x, "response")
                     )

 # Reload export object
 export_object$reload()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Export$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Export$print`
## ------------------------------------------------

## Not run:
# x is API response when export is requested
export_object <- Export$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print export object
 export_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Export$reload`
## ------------------------------------------------

## Not run:
# x is API response when export is requested
export_object <- Export$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload export object
```

```
export_object$reload()

## End(Not run)
```

Exports *R6 Class representing storage exports endpoints*

Description

R6 Class representing storage exports resource endpoints.

Super class

sevenbridges2::Resource -> Exports

Public fields

URL List of URL endpoints for this resource.

Methods

Public methods:

- Exports$new()
- Exports$query()
- Exports$get()
- Exports$submit_export()
- Exports$delete()
- Exports$bulk_get()
- Exports$bulk_submit_export()
- Exports$clone()

Method new(): Create a new Exports object.

Usage:

```
Exports$new(...)
```

Arguments:

... Other response arguments.

Method query(): This call lists export jobs initiated by a particular user. Note that when you export a file from a project on the Platform into a volume, you write to your cloud storage bucket.

Usage:

```
Exports$query(
  volume = NULL,
  state = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`volume` Volume id or Volume object. List all exports into this particular volume. Optional.

`state` The state of the export job. Possible values are:

- `PENDING`: the export is queued;
- `RUNNING`: the export is running;
- `COMPLETED`: the export has completed successfully;
- `FAILED`: the export has failed.

Example:

```
state = c("RUNNING", "FAILED")
```

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `Export` objects.

Examples:

```
\dontrun{
 exports_object <- Exports$new(
   auth = auth
 )

 # List all your running or failed export jobs on the volume
 exports_object$query(volume = volume, state = c("RUNNING", "FAILED"))
}
```

Method `get()`**:** This call will return the details of an export job.

Usage:

```
Exports$get(id, ...)
```

Arguments:

`id` The export job identifier (id).

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Export` object.

Examples:

```
\dontrun{
 exports_object <- Exports$new(
   auth = auth
 )

 # Get export job by ID
 exports_object$get(id = id)
}
```

Method `submit_export()`**:** This call lets you queue a job to export a file from a project on the Platform into a volume. The file selected for export must not be a public file or an alias. Aliases are objects stored in your cloud storage bucket which have been made available on the Platform. The volume you are exporting to must be configured for read-write access. To do this, set the `access_mode` parameter to `RW` when creating or modifying a volume.

Essentially, the call writes to your cloud storage bucket via the volume. If this call is successful, the original project file will become an alias to the newly exported object on the volume. The source file will be deleted from the Platform and, if no more copies of this file exist, it will no longer count towards your total storage price on the Platform.
In summary, once you export a file from the Platform to a volume, it is no longer part of the storage on the Platform and cannot be exported again.

Read more about this operation in our documentation here.

If you want to export multiple files, the recommended way is to do it in bulk considering the API rate limit (learn more). Bulk operations will be implemented in next releases.

Usage:

```
Exports$submit_export(
  source_file,
  destination_volume,
  destination_location,
  overwrite = FALSE,
  copy_only = FALSE,
  properties = NULL,
  ...
)
```

Arguments:

`source_file` File id or File object you want to export to the volume.

`destination_volume` Volume id or Volume object you want to export files into.

`destination_location` Volume-specific location to which the file will be exported. This location should be recognizable to the underlying cloud service as a valid key or path to a new file. Please note that if this volume has been configured with a prefix parameter, the value of prefix will be prepended to location before attempting to create the file on the volume.
If you would like to export the file into a folder on the volume, please add the folder name as a prefix before the file name in the form `<folder-name>/<file-name>`.

`overwrite` Set to `TRUE` if you want to overwrite the item if another one with the same name already exists at the destination.

`copy_only` If `TRUE`, file will be copied to a volume but source file will remain on the Platform.

`properties` Named list of additional volume properties, like:

- `sse_algorithm` - S3 server-side encryption to use when exporting to this bucket. Supported values: `AES256` (SSE-S3 encryption), `aws:kms`, `null` (no server-side encryption). Default: `AES256`.
- `sse_aws_kms_key_Id`: Applies to type: `s3`. If AWS KMS encryption is used, this should be set to the required KMS key. If not set and `aws:kms` is set as `sse_algorithm`, default KMS key is used.
- `aws_canned_acl`: S3 canned ACL to apply on the object on during export. Supported values: any one of S3 canned ACLs; `null` (do not apply canned ACLs). Default: `null`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Export` object.

Examples:

```
\dontrun{
 exports_object <- Exports$new(
   auth = auth
 )

 # Submit export job
 exp_job1 <- exports_object$submit_export(
                           source_file = test_file,
                           destination_volume = vol1,
                           destination_location = "new_volume_file.txt"
                 )
}
```

Method `delete()`**:** Export jobs cannot be deleted.

Usage:

```
Exports$delete()
```

Method `bulk_get()`**:** This call returns the details of a bulk export job. When you export files from a project on the Platform into a volume, you write to your cloud storage bucket. This call obtains the details of that job.

Usage:

```
Exports$bulk_get(exports)
```

Arguments:

`exports` The list of the export job IDs as returned by the call to start a bulk export job or list of `Export` objects.

Returns: `Collection` with list of `Export` objects.

Examples:

```
\dontrun{
 exports_object <- Exports$new(
                    auth = auth,
                   )

 # List export jobs
 exports_object$bulk_get(
  exports = list("export-job-id-1", "export-job-id-2")
  )
}
```

Method `bulk_submit_export()`**:** Bulk export files from your project on the Seven Bridges Platform into your volume. One call can contain up to 100 items. Files selected for export must not be public files or aliases. Aliases are objects stored in your cloud storage bucket which have been made available on the Platform. The volume you are exporting to must be configured for read-write access. To do this, set the `access_mode` parameter to RW when creating or modifying a volume.

Essentially, the call writes to your cloud storage bucket via the volume. If this call is successful, the original project files will become aliases to the newly exported objects on the volume. Source files will be deleted from the Platform and, if no more copies of the files exist, they will no longer count towards your total storage price on the Platform. In summary, once you export files from the Platform to a volume, they are no longer part of the storage on the Platform and cannot be exported again.

Learn more about using the Volumes API for Amazon S3 and for Google Cloud Storage.

Usage:

```
Exports$bulk_submit_export(items, copy_only = FALSE)
```

Arguments:

`items` Nested list of elements containing information about each file to be exported. For each element, users must provide:

- `source_file` - File ID or File object you want to export to the volume,
- `destination_volume` - Volume ID or Volume object you want to export files into.
- `destination_location` - Volume-specific location to which the file will be exported. This location should be recognizable to the underlying cloud service as a valid key or path to a new file. Please note that if this volume has been configured with a `prefix` parameter, the value of `prefix` will be prepended to the location before attempting to create the file on the volume.
  If you would like to export the file into a folder on the volume, please add folder name as a prefix before the file name in the `<folder-name>/<file-name>` form.
- `overwrite` - Set to `TRUE` if you want to overwrite the item with the same name if it already exists at the destination.
- `properties` - Named list of additional volume properties, like:
  - `sse_algorithm` - S3 server-side encryption to use when exporting to this bucket. Supported values: `AES256` (SSE-S3 encryption), `aws:kms`, `null` (no server-side encryption). Default: `AES256`.

- `sse_aws_kms_key_Id`: Applies to type: `s3`. If AWS KMS encryption is used, this should be set to the required KMS key. If not set and `aws:kms` is set as `sse_algorithm`, default KMS key is used.
- `aws_canned_acl`: S3 canned ACL to apply on the object during export. Supported values: any one of S3 canned ACLs; `null` (do not apply canned ACLs). Default: `null`.

Example of the list:

```
items <- list(
          list(
            source_file = "test_file-id",
            destination_volume = "volume-id",
            destination_location = "new_volume_file.txt"
          ),
          list(
            source_file = "test_file_obj",
            destination_volume = "test_volume_obj",
            destination_location = "/volume_folder/exported_file.txt",
            overwrite = TRUE
          ),
          list(
            source_file = "project_file_3_id",
            destination_volume = "volume-id",
            destination_location = "project_file_3.txt",
            properties = list(
              sse_algorithm = "AES256"
            )
          )
        )
```

Read more on how to export files from your project to a volume or a volume folder.
Utility function `prepare_items_for_bulk_export` can help you prepare the `items` parameter for the `bulk_submit_export()` method.

`copy_only` If set to true, the files will be copied to a volume but the source files will remain on the Platform.

Returns: `Collection` with list of `Export` objects.

Examples:

```
\dontrun{
 exports_object <- Exports$new(
                    auth = auth
                   )

 # Submit new bulk export into a volume
 exports_object$bulk_submit_export(items = list(
   list(
     source_file = "test_file-id",
     destination_volume = "volume-id",
     destination_location = "new_volume_file.txt"
   ),
```

```
    list(
      source_file = test_file_obj,
      destination_volume = test_volume_obj,
      destination_location = "/volume_folder/exported_file.txt",
      overwrite = TRUE
    ),
    list(
      source_file = "project_file_3_id",
      destination_volume = "volume-id",
      destination_location = "project_file_3.txt",
      properties = list(
       sse_algorithm = "AES256"
      )
    )
   ), copy_only = TRUE
  )
 }
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Exports$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Exports$query`
## ------------------------------------------------

## Not run:
 exports_object <- Exports$new(
   auth = auth
 )

 # List all your running or failed export jobs on the volume
 exports_object$query(volume = volume, state = c("RUNNING", "FAILED"))

## End(Not run)


## ------------------------------------------------
## Method `Exports$get`
## ------------------------------------------------

## Not run:
 exports_object <- Exports$new(
   auth = auth
 )
```

```
 # Get export job by ID
 exports_object$get(id = id)

## End(Not run)


## ------------------------------------------------
## Method `Exports$submit_export`
## ------------------------------------------------

## Not run:
 exports_object <- Exports$new(
   auth = auth
 )

 # Submit export job
 exp_job1 <- exports_object$submit_export(
                          source_file = test_file,
                          destination_volume = vol1,
                          destination_location = "new_volume_file.txt"
               )

## End(Not run)


## ------------------------------------------------
## Method `Exports$bulk_get`
## ------------------------------------------------

## Not run:
 exports_object <- Exports$new(
                    auth = auth,
                   )

 # List export jobs
 exports_object$bulk_get(
  exports = list("export-job-id-1", "export-job-id-2")
  )

## End(Not run)


## ------------------------------------------------
## Method `Exports$bulk_submit_export`
## ------------------------------------------------

## Not run:
 exports_object <- Exports$new(
                    auth = auth
                   )

 # Submit new bulk export into a volume
```

```
exports_object$bulk_submit_export(items = list(
  list(
    source_file = "test_file-id",
    destination_volume = "volume-id",
    destination_location = "new_volume_file.txt"
  ),
  list(
    source_file = test_file_obj,
    destination_volume = test_volume_obj,
    destination_location = "/volume_folder/exported_file.txt",
    overwrite = TRUE
  ),
  list(
    source_file = "project_file_3_id",
    destination_volume = "volume-id",
    destination_location = "project_file_3.txt",
    properties = list(
     sse_algorithm = "AES256"
    )
  )
 ), copy_only = TRUE
)

## End(Not run)
```

`File` *R6 Class representing a File*

Description

R6 Class representing a resource for managing files and folders.

Super class

`sevenbridges2::Item` -> `File`

Public fields

`URL` List of URL endpoints for this resource.

`id` File ID.

`name` File name.

`size` File size.

`project` Project ID if any, where file/folder is located.

`created_on` Date file/folder was created on.

`modified_on` Date file/folder was modified on.

`storage` File/folder's storage type.

`origin` Task ID if file/folder is produced by some task execution.

`tags` List of tags associated with the file.

`metadata` List of metadata associated with the file.

`url` File download URL.

`parent` Parent folder ID.

`type` This can be of type `file` or `folder`.

`secondary_files` Secondary files linked to the file, if they exist.

Methods

Public methods:

- `File$new()`
- `File$print()`
- `File$detailed_print()`
- `File$reload()`
- `File$update()`
- `File$add_tag()`
- `File$copy_to()`
- `File$get_download_url()`
- `File$get_metadata()`
- `File$set_metadata()`
- `File$move_to_folder()`
- `File$list_contents()`
- `File$delete()`
- `File$download()`
- `File$submit_export()`
- `File$clone()`

Method `new()`**:** Create a new File object.

Usage:

```
File$new(res = NA, ...)
```

Arguments:

`res` Response containing File object information.

`...` Other response arguments.

Method `print()`**:** Print method for File class.

Usage:

```
File$print()
```

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print file object
 file_object$print()
}
```

Method `detailed_print()`**:** Detailed print method for File class.

Usage:

```
File$detailed_print()
```

Details: The call returns the file's name, its tags, and all of its metadata. Apart from regular file fields there are some additional fields:

- `storage` field denotes the type of storage for the file which can be either PLATFORM or VOLUME depending on where the file is stored.
- `origin` field denotes the task that produced the file, if it was created by a task on the Seven Bridges Platform.
- `metadata` field lists the metadata fields and values for the file.
- `tags` field lists the tags for the file. Learn more about tagging your files on the Platform.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Detailed print of file object
 file_object$detailed_print()
}
```

Method `reload()`**:** Reload File object information.

Usage:

```
File$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `File` object.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload file object
 file_object$reload()
}
```

Method `update()`**:** Updates the name, the full set of metadata, and tags for a specified file.

Usage:

```
File$update(name = NULL, metadata = NULL, tags = NULL, ...)
```

Arguments:

`name` The new name of the file.

`metadata` The metadata fields and their values that you want to update. This is a named list of key-value pairs. The keys and values are strings.

`tags` The tags you want to update, represented as unnamed list of values to add as tags.

`...` Other arguments that can be passed to core `api()` function like 'limit', 'offset', 'fields', etc.

Returns: Updated `File` object.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Update file object
 file_object$update(name = "new_name")
}
```

Method `add_tag()`**:** This method allows you to tag files on the Platform. You can tag your files on the Platform with keywords to make it easier to identify and organize files you've imported from public datasets or copied between projects.
More details on how to use this call can be found here.

Usage:

```
File$add_tag(tags, overwrite = FALSE, ...)
```

Arguments:

`tags` The tags you want to update, represented as unnamed list of values to add as tags.

`overwrite` Set to `TRUE` if you want to overwrite existing tags. Default: `FALSE`.

`...` Additional parameters that can be passed to the method.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add new tag to file object
 file_object$add_tag(tags = c("new_tag"))
}
```

Method `copy_to()`**:** This call copies the specified file to a new project. Files retain their metadata when copied, but may be assigned new names in their target project. To make this call, you should have copy permission within the project you are copying from.
Note: If you want to copy multiple files, the recommended way is to do it in bulk considering the API rate limit (learn more). You can do that using `Auth$copy_files()` operation.

Usage:

```
File$copy_to(project, name = NULL, ...)
```

Arguments:

`project` The ID of the project or a Project object where you want to copy the file to.

`name` The new name the file will have in the target project. If its name will not change, omit this key.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: Copied `File` object.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Copy file object to project
 file_object$copy_to(project = project)
```

```
}
```

Method `get_download_url()`**:** This method returns a URL that you can use to download the specified file.

Usage:

```
File$get_download_url(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get download url for file object
 file_object$get_download_url()
}
```

Method `get_metadata()`**:** This call returns the metadata values for the specified file.

Usage:

```
File$get_metadata(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get metadata for file object
 file_object$get_metadata()
}
```

Method `set_metadata()`**:** This call changes the metadata values for the specified file. More details about how to modify metadata, you can find in the API documentation.

Usage:

```
File$set_metadata(metadata_fields, overwrite = FALSE, ...)
```

Arguments:

`metadata_fields` Enter a list of key-value pairs of metadata fields and metadata values.

`overwrite` Set to `TRUE` if you want to overwrite existing tags. Default: `FALSE`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Set metadata for file object
 file_object$set_metadata(metadata_fields = list("field_1" = "value_1"))
}
```

Method `move_to_folder()`**:** This call moves a file from one folder to another. Moving of files is only allowed within the same project.

Usage:

```
File$move_to_folder(parent, name = NULL)
```

Arguments:

`parent` The ID of target folder or a File object which must be of type `FOLDER`.

`name` Specify a new name for a file in case you want to rename it. If you want to use the same name, omit this key.

Returns: Moved `File` object.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Move file object to a project
 file_object$move_to_folder(parent = "parent-folder-id")
}
```

Method `list_contents()`**:** List folder contents.

Usage:

```
File$list_contents(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `File` objects.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List folder's content
 file_object$list_contents()
}
```

Method `delete()`**:** Delete method for File objects.

Usage:

```
File$delete()
```

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete file object
 file_object$delete()
}
```

Method `download()`**:** Download method for File objects. It allows downloading a platform file to your local computer. To specify the destination for your download, you should provide the path to the destination directory as `directory_path` parameter.

Usage:

```
File$download(
  directory_path,
  filename = self$name,
  method = "curl",
  retry_count = getOption("sevenbridges2")$default_retry_count,
  retry_timeout = getOption("sevenbridges2")$default_retry_timeout
)
```

Arguments:

`directory_path` Path to the destination directory of a new file.

`filename` Full name for the new file, including its extension. By default, the name field of File object will be used.

`method` Method to be used for downloading files. By default, this parameter is set to `curl`.

`retry_count` Number of retries if error occurs during download.

`retry_timeout` Number of seconds between two retries.

Examples:

```
\dontrun{
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Download file object
 file_object$download(directory_path = ".")
}
```

Method `submit_export()`**:** This call lets you queue a job to export this file from a project on the Platform into a volume. The file selected for export must not be a public file or an alias. Aliases are objects stored in your cloud storage bucket which have been made available on the Platform. The volume you are exporting to must be configured for read-write access. To do this, set the `access_mode` parameter to `RW` when creating or modifying a volume.

Essentially, the call writes to your cloud storage bucket via the volume. If this call is successful, the original project file will become an alias to the newly exported object on the volume. The source file will be deleted from the Platform and, if no more copies of this file exist, it will no longer count towards your total storage price on the Platform.
In summary, once you export a file from the Platform to a volume, it is no longer part of the storage on the Platform and cannot be exported again.

Read more about this operation in our documentation here.
If you want to export multiple files, the recommended way is to do it in bulk considering the API rate limit (learn more) (bulk operations will be implemented in next releases).

Usage:

```
File$submit_export(
  destination_volume,
  destination_location,
  overwrite = FALSE,
  copy_only = FALSE,
  properties = NULL,
  ...
)
```

Arguments:

`destination_volume` Volume id or Volume object you want to export files into. Required.

`destination_location` Volume-specific location to which the file will be exported. This location should be recognizable to the underlying cloud service as a valid key or path to a new file. Please note that if this volume has been configured with a prefix parameter, the value of prefix will be prepended to location before attempting to create the file on the volume. If you would like to export the file into some folder on the volume, please add folder name as prefix before file name in form `<folder-name>/<file-name>`.

`overwrite` Set to `TRUE` if you want to overwrite the item that already exists at the destination. Default: `FALSE`.

`copy_only` If `TRUE`, file will be copied to a volume but source file will remain on the Platform.

`properties` Named list of additional volume properties, like:

- `sse_algorithm` - S3 server-side encryption to use when exporting to this bucket. Supported values: `AES256` (SSE-S3 encryption), `aws:kms`, `null` (no server-side encryption). Default: `AES256`.
- `sse_aws_kms_key_Id`: Applies to type: `s3`. If AWS KMS encryption is used, this should be set to the required KMS key. If not set and `aws:kms` is set as `sse_algorithm`, default KMS key is used.
- `aws_canned_acl`: S3 canned ACL to apply on the object on during export. Supported values: any one of S3 canned ACLs; `null` (do not apply canned ACLs). Default: `null`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Export` object.

Examples:

```
\dontrun{
 # x is API response when file is requested
 file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Export file object to a volume
```

```
 file_object$submit_export(
               destination_volume = volume,
               destination_location = location
             )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
File$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `File$print`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                   res = x,
                   href = x$href,
                   auth = auth,
                   response = attr(x, "response")
                  )

 # Print file object
 file_object$print()

## End(Not run)


## ------------------------------------------------
## Method `File$detailed_print`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                   res = x,
                   href = x$href,
                   auth = auth,
                   response = attr(x, "response")
                  )

 # Detailed print of file object
 file_object$detailed_print()

## End(Not run)
```

```
## ------------------------------------------------
## Method `File$reload`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload file object
 file_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `File$update`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Update file object
 file_object$update(name = "new_name")

## End(Not run)


## ------------------------------------------------
## Method `File$add_tag`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
```

```
 # Add new tag to file object
 file_object$add_tag(tags = c("new_tag"))

## End(Not run)


## ------------------------------------------------
## Method `File$copy_to`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Copy file object to project
 file_object$copy_to(project = project)

## End(Not run)


## ------------------------------------------------
## Method `File$get_download_url`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get download url for file object
 file_object$get_download_url()

## End(Not run)


## ------------------------------------------------
## Method `File$get_metadata`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
```

```
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get metadata for file object
 file_object$get_metadata()

## End(Not run)


## ------------------------------------------------
## Method `File$set_metadata`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Set metadata for file object
 file_object$set_metadata(metadata_fields = list("field_1" = "value_1"))

## End(Not run)


## ------------------------------------------------
## Method `File$move_to_folder`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Move file object to a project
 file_object$move_to_folder(parent = "parent-folder-id")

## End(Not run)


## ------------------------------------------------
## Method `File$list_contents`
## ------------------------------------------------

## Not run:
```

```
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List folder's content
 file_object$list_contents()

## End(Not run)


## ------------------------------------------------
## Method `File$delete`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete file object
 file_object$delete()

## End(Not run)


## ------------------------------------------------
## Method `File$download`
## ------------------------------------------------

## Not run:
# x is API response when file is requested
file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Download file object
 file_object$download(directory_path = ".")

## End(Not run)


## ------------------------------------------------
```

```
## Method `File$submit_export`
## ------------------------------------------------

## Not run:
 # x is API response when file is requested
 file_object <- File$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Export file object to a volume
 file_object$submit_export(
               destination_volume = volume,
               destination_location = location
             )

## End(Not run)
```

`Files` *R6 Class representing files endpoints.*

Description

R6 Class representing Files resource.

Super class

`sevenbridges2::Resource` -> `Files`

Public fields

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Files$new()`
- `Files$query()`
- `Files$get()`
- `Files$delete()`
- `Files$copy()`
- `Files$create_folder()`
- `Files$bulk_delete()`
- `Files$bulk_get()`

- Files$bulk_update()
- Files$bulk_edit()
- Files$async_bulk_copy()
- Files$async_bulk_delete()
- Files$async_bulk_move()
- Files$async_get_copy_job()
- Files$async_get_delete_job()
- Files$async_get_move_job()
- Files$async_list_file_jobs()
- Files$clone()

Method `new()`**:** Create new Files resource object.

Usage:

```
Files$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** This call returns a list of files and subdirectories in a specified project or directory within a project, with specified properties that you can access. The project or directory whose contents you want to list is specified as a query parameter in the call. Further properties to filter by can also be specified as query parameters.

Note that this call lists both files and subdirectories in the specified project or directory within a project, but not the contents of the subdirectories.
To list the contents of a subdirectory, make a new call and specify the subdirectory ID as the `parent` parameter.
For more details, see our API documentation.

Usage:

```
Files$query(
  project = NULL,
  parent = NULL,
  name = NULL,
  metadata = NULL,
  origin = NULL,
  tag = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`project` Project identifier (ID) as string or a Project object. Project should not be used together with parent. If parent is used, the call will list the content of the specified folder, within the project to which the folder belongs. If project is used, the call will list the content at the root of the project's files.

`parent` The parent folder identifier as string or a File object which must be of type `FOLDER`. Should not be used together with project. If parent is used, the call will list the content of the specified folder, within the project to which the folder belongs. If project is used, the call will list the content at the root of the project's files.

`name` Name of the file. List files with this name. Note that the name must be an exact complete string for the results to match. Multiple names can be represented as a vector.

`metadata` List file with this metadata field values. List only files that have the specified value in metadata field. Different metadata fields are represented as a named list. You can also define multiple instances of the same metadata field.

`origin` Task object. List only files produced by task.

`tag` Filters the files based on the specified tag(s). Each tag must be an exact, complete match, for the results to match. Tags may include spaces. Multiple tags should be provided as a vector of strings. The method will return files that have any of the specified tags.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function as 'fields', etc.

Returns: `Collection` of `File` objects.

Examples:

```
\dontrun{
 files_object <- Files$new(auth = auth)

 # Query files in a project
 files_object$query(project = project)
}
```

Method `get()`**:** This call returns a single File object with its details. The call returns the file's name, its tags, and all of its metadata. Files are specified by their IDs, which you can obtain by making the API call to list all files in a project.

Usage:

```
Files$get(id, ...)
```

Arguments:

`id` The file ID.

`...` Other arguments that can be passed to core `api()` function as 'fields', etc.

Returns: `File` object.

Examples:

```
\dontrun{
 files_object <- Files$new(auth = auth)

 # Get file using id
 files_object$get(id = id)
}
```

Method `delete()`**:** This call removes a file from the Seven Bridges Platform. Files are specified by their IDs, which you can obtain by using `Files$query()` to list files or by getting a single file using `Files$get()`.

Usage:

```
Files$delete(file, ...)
```

Arguments:

`file` `File` object or file ID.

`...` Other arguments that can be passed to core `api()` function as 'fields', etc.

Examples:

```
\dontrun{
 files_object <- Files$new(auth = auth)

 # Delete a file
 files_object$delete(file = file)
}
```

Method `copy()`**:** Copy file/files to the specified project. This call allows you to copy files between projects. Unlike the call to copy a file between projects, this call lets you batch the copy operation and copy a list of files at a time.
More information can be found here here.

Usage:

```
Files$copy(files, destination_project)
```

Arguments:

`files` The list of files' IDs or list of File object to copy.

`destination_project` Project object or project ID. where you want to copy files into.

Examples:

```
\dontrun{
 files_object <- Files$new(auth = auth)

 # Copy files to a project
 files_object$copy(
               file = file,
               destination_project = project
              )
}
```

Method `create_folder()`**:** A method for creating a new folder. It allows you to create a new folder on the Platform within the root folder of a specified project or the provided parent folder. Remember that you should provide either the destination project (as the `project` parameter) or the destination folder (as the `parent` parameter), not both.
More information you may find here.

Usage:

```
Files$create_folder(name, parent = NULL, project = NULL)
```

Arguments:

`name` The name of the folder you are about to create.

`parent` The ID of the parent destination folder or a File object which must be of type `FOLDER`.

`project` The ID of the destination project, or a Project object.

Examples:

```
\dontrun{
 files_object <- Files$new(auth = auth)

 # Create folder in a project
 files_object$create_folder(
                name = name,
                project = project
               )
}
```

Method `bulk_delete()`**:** This method facilitates bulk file deletion. It accepts either a list of `File` objects or a list containing files' IDs.

Usage:

```
Files$bulk_delete(files)
```

Arguments:

`files` Either a list of `File` objects or a list of strings (IDs) representing the files you intend to delete.

Returns: None. The function only displays the IDs of the deleted files in the console.

Examples:

```
\dontrun{
 # Delete two files by providing their IDs
 a$files$bulk_delete(files = list("file_1_ID", "file_2_ID"))
}

\dontrun{
 # Delete two files by providing a list of File objects
 a$files$bulk_delete(files = list(File_Object_1, File_Object_2))
}
```

Method `bulk_get()`**:** This call returns the details of multiple specified files, including file names and file metadata. The maximum number of files you can retrieve the details for per call is 100.

Usage:

```
Files$bulk_get(files)
```

Arguments:

`files` A list of `File` objects or list of strings (IDs) of the files you are querying for details.

Returns: `Collection` (list of `File` objects).

Examples:

```
\dontrun{
 # Get details of multiple files
 a$files$bulk_get(
               files = list("file_1_ID", "file_2_ID")
              )
}
```

Method `bulk_update()`**:** A method that sets new information for specified files, replacing all existing information and erasing omitted parameters.

Usage:

```
Files$bulk_update(files)
```

Arguments:

`files` List of `File` objects.

Details: For each of the specified files, the call sets a new `name`, new `tags`, and `metadata`. When editing fields in the `File` objects you wish to update, keep the following in mind:

- The `name` field should be a string representing the new name of the file.
- The `metadata` field should be a named list of key-value pairs. The keys and values should be strings.
- The `tags` field should be an unnamed list of values.

The maximum number of files you can update the details for per call is 100.

Returns: `Collection` (list of `File` objects).

Examples:

```
\dontrun{
 # Update details of multiple files
 a$files$bulk_update(
               files = list(File_Object_1, File_Object_2)
              )
}
```

Method `bulk_edit()`**:** This method modifies the existing information for specified files or adds new information while preserving omitted parameters.

Usage:

```
Files$bulk_edit(files)
```

Arguments:

`files` List of `File` objects.

Details: For each of the specified files, the call edits its `name`, `tags`, and `metadata`. When editing fields in the `File` objects you wish to update, keep the following in mind:

- The `name` field should be a string representing the new name of the file.
- The `metadata` field should be a named list of key-value pairs. The keys and values should be strings.
- The `tags` field should be an unnamed list of values.

The maximum number of files you can update the details for per call is 100.

Returns: `Collection` (list of `File` objects).

Examples:

```
\dontrun{
 # Edit details of multiple files
 a$files$bulk_edit(
              files = list(File_Object_1, File_Object_2)
             )
}
```

Method `async_bulk_copy()`**:** This call lets you perform a bulk copy of files and folders. Any underlying folder structure will be preserved. You can copy:

- to a folder within the same project,
- to another project,
- to a folder in another project.

Usage:

```
Files$async_bulk_copy(items)
```

Arguments:

`items` Nested list of elements containing information about each file/folder to be copied. For each element, you must provide:

- `file` - The ID of the file or folder you are copying. Copying the project root folder is not allowed. Use the API call for listing all files to obtain the ID.
- `parent` - The ID of the folder you are copying files to. It should not be used together with `project`. If `project` is used, the items will be imported to the root of the project files. If `parent` is used, the import will take place into the specified folder, within the project to which the folder belongs.
- `project` - The project you are copying the file to. It should not be used together with `parent`. If `parent` is used, the import will take place into the specified folder, within the project to which the folder belongs. If `project` is used, the items will be imported to the root of the project files.
- `name` - Enter the new name for the file if you want to rename it in the destination folder.

Example of the list:

```
items <- list(
          list(
            file = '<file-id-1>',
            parent = '<folder-id>'
          ),
          list(
            file = '<file-id-2>',
            project = '<project-id-1>',
            name = 'copied_file.txt'
          ),
          list(
            file = '<file-id-3>',
```

```
              parent = '<parent-id-2>',
              name = 'copied_file2.txt'
            )
          )
```

Read more on how to perform async copy action on multiple files.

Returns: AsyncJob object.

Examples:

```
\dontrun{
 # Copy multiple files
 a$files$async_bulk_copy(
   items = list(
           list(
             file = '<file-id-1>',
             parent = '<folder-id>'
           ),
           list(
             file = '<file-id-2>',
             project = '<project-id-1>',
             name = 'copied_file.txt'
           ),
           list(
             file = '<file-id-3>',
             parent = '<parent-id-2>',
             name = 'copied_file2.txt'
           )
         )
  )
}
```

Method `async_bulk_delete()`**:** This call lets you perform an asynchronous bulk deletion of files or folders. Deleting folders which aren't empty is allowed.

Usage:

```
Files$async_bulk_delete(items)
```

Arguments:

`items` List of File objects (both `file` or `folder` type) or list of IDs of files/folders you want to delete. Read more on how to perform async delete action on multiple files.

Returns: AsyncJob object.

Examples:

```
\dontrun{
 # Delete multiple files
 a$files$async_bulk_delete(
   items = list(file_obj1, file_obj2, "<folder-id-string>", "<file-id>")
 )
}
```

Method `async_bulk_move()`**:** This call lets you perform a bulk move operation of files and folders. You can move:

- to a root project folder,
- to a subfolder within the same project or a different project.

Usage:

```
Files$async_bulk_move(items)
```

Arguments:

`items` Nested list of elements containing information about each file/folder to be moved. For each element, you must provide:

- `file` - The ID of the file or folder you are moving. Use the API call for listing all files or folders to obtain the ID.
- `parent` - The ID of the folder you are moving the files to, which should not be used along with `project`. If `project` is used, the items will be imported to the root of the project files. If `parent` is used, the import will take place into the specified folder, within the project to which the folder belongs.
- `project` - The project you are moving the files to. It should not be used together with `parent`. If `parent` is used, the import will take place into the specified folder, within the project to which the folder belongs. If `project` is used, the items will be imported to the root of the project files.
- `name` - Enter the new name for the file or folder if you want to rename at the destination.

Example of the list:

```
items <- list(
          list(
            file = '<file-id-1>',
            parent = '<folder-id>'
          ),
          list(
            file = '<file-id-2>',
            project = '<project-id-1>',
            name = 'moved_file.txt'
          ),
          list(
            file = '<file-id-3>',
            parent = '<parent-id-2>',
            name = 'moved_file2.txt'
          )
        )
```

Read more on how to perform async move action on multiple files.

Details: Rules for moving files and folders:

- The file ID is preserved after the move.
- The folder ID is changed after the move.
- The destination must be an existing folder.
- If the target folder contains a folder with the same name, the contents of both folders will be merged.

- If a file with the same name already exists, the source file will be automatically renamed (by adding a numeric prefix).
- You need to have WRITE permissions for both source and destination folders.

Returns: AsyncJob object.

Examples:

```
\dontrun{
 # Move multiple files
 a$files$async_bulk_move(
   items = list(
           list(
             file = '<file-id-1>',
             parent = '<folder-id>'
           ),
           list(
             file = '<file-id-2>',
             project = '<project-id-1>',
             name = 'moved_file.txt'
           ),
           list(
             file = '<file-id-3>',
             parent = '<parent-id-2>',
             name = 'moved_file2.txt'
           )
         )
  )
}
```

Method `async_get_copy_job()`**:** This call retrieves the details of an asynchronous bulk copy job. This information will be available for up to a month after the job has been completed.

Usage:

```
Files$async_get_copy_job(job_id)
```

Arguments:

`job_id` The ID of the copy job you are querying. This ID can be found within the API response for the call for copying files. The function also accepts an AsyncJob object and extracts the ID.

Returns: AsyncJob object.

Examples:

```
\dontrun{
 # Get details of an async copy job
 a$files$async_get_copy_job(job_id = "job-id")
}
```

Method `async_get_delete_job()`**:** This call retrieves the details of an asynchronous bulk deletion job. This information will be available for up to a month after the job has been completed.

Usage:

```
Files$async_get_delete_job(job_id)
```

Arguments:

`job_id` The ID of the delete job you are querying. This ID can be found within the API response for the call for deleting files. The function also accepts an AsyncJob object and extracts the ID.

Returns: `AsyncJob` object.

Examples:

```
\dontrun{
 # Get details of an async delete job
 a$files$async_get_delete_job(job_id = "job-id")
}
```

Method `async_get_move_job()`**:** This call retrieves the details of an asynchronous bulk move job. This information will be available for up to a month after the job has been completed.

Usage:

```
Files$async_get_move_job(job_id)
```

Arguments:

`job_id` The ID of the move job you are querying. This ID can be found within the API response for the call for moving files. The function also accepts an AsyncJob object and extracts the ID.

Returns: An `AsyncJob` object containing details of the move job.

Examples:

```
\dontrun{
 # Get details of an async move job
 a$files$async_get_move_job(job_id = "job-id")
}
```

Method `async_list_file_jobs()`**:** This call retrieves the details for all asynchronous bulk jobs you have started. This information will be available for up to a month after the job has been completed.

Usage:

```
Files$async_list_file_jobs(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

Returns: A `Collection` object containing a list of `AsyncJob` objects.

Examples:

```
\dontrun{
 # Get details of the first 5 async jobs
 a$files$async_list_file_jobs(limit = 5)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Files$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Files$query`
## ------------------------------------------------

## Not run:
 files_object <- Files$new(auth = auth)

 # Query files in a project
 files_object$query(project = project)

## End(Not run)


## ------------------------------------------------
## Method `Files$get`
## ------------------------------------------------

## Not run:
 files_object <- Files$new(auth = auth)

 # Get file using id
 files_object$get(id = id)

## End(Not run)


## ------------------------------------------------
## Method `Files$delete`
## ------------------------------------------------

## Not run:
 files_object <- Files$new(auth = auth)

 # Delete a file
```

```
 files_object$delete(file = file)

## End(Not run)


## ------------------------------------------------
## Method `Files$copy`
## ------------------------------------------------

## Not run:
 files_object <- Files$new(auth = auth)

 # Copy files to a project
 files_object$copy(
               file = file,
               destination_project = project
              )

## End(Not run)


## ------------------------------------------------
## Method `Files$create_folder`
## ------------------------------------------------

## Not run:
 files_object <- Files$new(auth = auth)

 # Create folder in a project
 files_object$create_folder(
                name = name,
                project = project
               )

## End(Not run)


## ------------------------------------------------
## Method `Files$bulk_delete`
## ------------------------------------------------

## Not run:
 # Delete two files by providing their IDs
 a$files$bulk_delete(files = list("file_1_ID", "file_2_ID"))

## End(Not run)

## Not run:
 # Delete two files by providing a list of File objects
 a$files$bulk_delete(files = list(File_Object_1, File_Object_2))

## End(Not run)
```

```
## ------------------------------------------------
## Method `Files$bulk_get`
## ------------------------------------------------

## Not run:
 # Get details of multiple files
 a$files$bulk_get(
               files = list("file_1_ID", "file_2_ID")
              )

## End(Not run)


## ------------------------------------------------
## Method `Files$bulk_update`
## ------------------------------------------------

## Not run:
 # Update details of multiple files
 a$files$bulk_update(
               files = list(File_Object_1, File_Object_2)
              )

## End(Not run)


## ------------------------------------------------
## Method `Files$bulk_edit`
## ------------------------------------------------

## Not run:
 # Edit details of multiple files
 a$files$bulk_edit(
               files = list(File_Object_1, File_Object_2)
              )

## End(Not run)


## ------------------------------------------------
## Method `Files$async_bulk_copy`
## ------------------------------------------------

## Not run:
 # Copy multiple files
 a$files$async_bulk_copy(
   items = list(
           list(
             file = '<file-id-1>',
             parent = '<folder-id>'
           ),
           list(
```

```
            file = '<file-id-2>',
            project = '<project-id-1>',
            name = 'copied_file.txt'
          ),
          list(
            file = '<file-id-3>',
            parent = '<parent-id-2>',
            name = 'copied_file2.txt'
          )
        )
  )

## End(Not run)


## ------------------------------------------------
## Method `Files$async_bulk_delete`
## ------------------------------------------------

## Not run:
 # Delete multiple files
 a$files$async_bulk_delete(
   items = list(file_obj1, file_obj2, "<folder-id-string>", "<file-id>")
 )

## End(Not run)


## ------------------------------------------------
## Method `Files$async_bulk_move`
## ------------------------------------------------

## Not run:
 # Move multiple files
 a$files$async_bulk_move(
   items = list(
          list(
            file = '<file-id-1>',
            parent = '<folder-id>'
          ),
          list(
            file = '<file-id-2>',
            project = '<project-id-1>',
            name = 'moved_file.txt'
          ),
          list(
            file = '<file-id-3>',
            parent = '<parent-id-2>',
            name = 'moved_file2.txt'
          )
        )
  )
```

```
## End(Not run)


## ------------------------------------------------
## Method `Files$async_get_copy_job`
## ------------------------------------------------

## Not run:
 # Get details of an async copy job
 a$files$async_get_copy_job(job_id = "job-id")

## End(Not run)


## ------------------------------------------------
## Method `Files$async_get_delete_job`
## ------------------------------------------------

## Not run:
 # Get details of an async delete job
 a$files$async_get_delete_job(job_id = "job-id")

## End(Not run)


## ------------------------------------------------
## Method `Files$async_get_move_job`
## ------------------------------------------------

## Not run:
 # Get details of an async move job
 a$files$async_get_move_job(job_id = "job-id")

## End(Not run)


## ------------------------------------------------
## Method `Files$async_list_file_jobs`
## ------------------------------------------------

## Not run:
 # Get details of the first 5 async jobs
 a$files$async_list_file_jobs(limit = 5)

## End(Not run)
```

Import *R6 Class representing an Import job*

Description

R6 Class representing a resource for managing volume import jobs.

Super class

`sevenbridges2::Item` -> `Import`

Public fields

`URL` List of URL endpoints for this resource.

`id` Import job string identifier.

`state` The state of the import job. Possible values are:

- `PENDING`: the import is queued;
- `RUNNING`: the import is running;
- `COMPLETED`: the import has completed successfully;
- `FAILED`: the import has failed.

`overwrite` Indicates whether the imported file or folder name was overwritten if another with the same name already existed.

`autorename` Indicates whether the imported file or folder name was automatically renamed (by prefixing its name with an underscore and number) if another with the same name already existed.

`preserve_folder_structure` Whether the imported folder structure was preserved or not.

`source` List containing source volume id and source location of the file/folder is being imported to the platform.

`destination` List containing the source volume ID and the source location of the file or folder being imported to the platform.

`started_on` Time when the import job started.

`finished_on` Time when the import job ended.

`error` In case of error in the import job, standard API error is returned here.

`result` File object that was imported.

Methods

Public methods:

- `Import$new()`
- `Import$print()`
- `Import$reload()`
- `Import$clone()`

Method `new()`**:** Create a new Import object.

Usage:

```
Import$new(res = NA, ...)
```

Arguments:

`res` Response containing Import object information.

`...` Other response arguments.

Method `print()`**:** Print method for Import class.

Usage:

```
Import$print()
```

Examples:

```
\dontrun{
 # x is API response when import is requested
 import_object <- Import$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print import object
 import_object$print()
}
```

Method `reload()`**:** Reload Import object information.

Usage:

```
Import$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Import` object.

Examples:

```
\dontrun{
 # x is API response when import is requested
 import_object <- Import$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload import object
 import_object$reload()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Import$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Import$print`
## ------------------------------------------------

## Not run:
 # x is API response when import is requested
 import_object <- Import$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print import object
 import_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Import$reload`
## ------------------------------------------------

## Not run:
 # x is API response when import is requested
 import_object <- Import$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload import object
 import_object$reload()

## End(Not run)
```

Imports *R6 Class representing storage imports endpoints*

Description

R6 Class for managing storage imports resource endpoints.

Super class

sevenbridges2::Resource -> Imports

Public fields

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Imports$new()`
- `Imports$query()`
- `Imports$get()`
- `Imports$submit_import()`
- `Imports$delete()`
- `Imports$bulk_get()`
- `Imports$bulk_submit_import()`
- `Imports$clone()`

Method `new()`**:** Create a new Imports object.

Usage:

```
Imports$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** This call lists import jobs initiated by a particular user. Note that when you import a file from your volume on your cloud storage provider (Amazon Web Services or Google Cloud Storage), you are creating an alias on the Platform which points to the file in your cloud storage bucket. Aliases appear as files on the Platform and can be copied, executed, and modified as such. They refer back to the respective file on the given volume.

Usage:

```
Imports$query(
  volume = NULL,
  project = NULL,
  state = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`volume` Volume id or Volume object. List all imports from this particular volume. Optional.

`project` Project id or Project object. List all volume imports to this particular project. Optional.

`state` The state of the import job. Possible values are:

- `PENDING`: the import is queued;
- `RUNNING`: the import is running;
- `COMPLETED`: the import has completed successfully;
- `FAILED`: the import has failed.

Example:

```
state = c("RUNNING", "FAILED")
```

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `Import` objects.

Examples:

```
\dontrun{
 imports_object <- Imports$new(
                    auth = auth
                   )

 # List import job
 imports_object$query()
}
```

Method `get()`**:** This call will return the details of an import job.

Usage:

```
Imports$get(id, ...)
```

Arguments:

`id` The import job identifier (id).

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Import` object.

Examples:

```
\dontrun{
 imports_object <- Imports$new(
                    auth = auth
                   )

 # List import job
 imports_object$get(id = id)
}
```

Method `submit_import()`**:** This call lets you queue a job to import a file or folder from a volume into a project on the Platform. Essentially, you are importing an item from your cloud storage provider (Amazon Web Services, Google Cloud Storage, Azure or Ali Cloud) via the volume onto the Platform.
If successful, an alias will be created on the Platform. Aliases appear on the Platform and can be copied, executed, and modified as such. They refer back to the respective item on the given volume.

If you want to import multiple files, the recommended way is to do it in bulk considering the API rate limit (learn more). Bulk operations will be implemented in next releases.

Usage:

```
Imports$submit_import(
  source_volume,
  source_location,
  destination_project = NULL,
  destination_parent = NULL,
  name = NULL,
  overwrite = FALSE,
  autorename = FALSE,
  preserve_folder_structure = NULL,
  ...
)
```

Arguments:

`source_volume` Volume id or Volume object you want to import files or folders from.

`source_location` File location name or folder prefix name on the volume you would like to import into some project/folder on the Platform.

`destination_project` Destination project id or Project object. Not required, but either `destination_project` or `destination_parent` directory must be provided.

`destination_parent` Folder id or File object (with `type = 'FOLDER'`). Not required, but either `destination_project` or `destination_parent` directory must be provided.

`name` The name of the alias to create. This name should be unique to the project.
If the name is already in use in the project, you should use the `overwrite` query parameter in this call to force any item with that name to be deleted before the alias is created. If name is omitted, the alias name will default to the last segment of the complete location (including the prefix) on the volume.

Segments are considered to be separated with forward slashes `/`. Allowed characters in file names are all alphanumeric and special characters except forward slash `/`, while folder names can contain alphanumeric and special characters `_`, `-` and `.`.

`overwrite` Set to `TRUE` if you want to overwrite the item if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder's content (files with the same name) will be overwritten, not the whole folder.

`autorename` Set to `TRUE` if you want to automatically rename the item (by prefixing its name with an underscore and number) if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder content will be renamed, not the whole folder.

`preserve_folder_structure` Set to `TRUE` if you want to keep the exact source folder structure. The default value is `TRUE` if the item being imported is a folder. Should not be used if you are importing a file. Bear in mind that if you use `preserve_folder_structure = FALSE`, the response will be the parent folder object containing imported files alongside with other files if they exist.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Import` object.

Examples:

```
\dontrun{
 imports_object <- Imports$new(
```

```
                    auth = auth
                   )

 # Submit new import into a project
 imports_object$submit_import(
  source_location = volume_file_object,
  destination_project = test_project_object,
  autorename = TRUE
 )
}
```

Method `delete()`**:** Import jobs cannot be deleted.

Usage:

```
Imports$delete()
```

Method `bulk_get()`**:** This call returns the details of a bulk import job. Note that when you import files from your volume on a cloud storage provider (Amazon Web Services or Google Cloud Storage), you create an alias on the Platform which points to the files in your cloud storage bucket. Aliases appear as files on the Platform and can be copied, executed, and modified.

Usage:

```
Imports$bulk_get(imports)
```

Arguments:

`imports` The list of the import job IDs as returned by the call to start a bulk import job or list of `Import` objects.

Returns: `Collection` with list of `Import` objects.

Examples:

```
\dontrun{
 imports_object <- Imports$new(
                    auth = auth
                   )

 # List import job
 imports_object$bulk_get(
  imports = list("import-job-id-1", "import-job-id-2")
  )
}
```

Method `bulk_submit_import()`**:** This call lets you perform a bulk import of files from your volume (either Amazon Web Services or Google Cloud Storage) into your project on the Platform. You can use this call to either import files to a specific folder or a project but you can also use it to import a folder and its files into another destination folder while preserving folder structure. One call can contain up to 100 items. Learn more about using the Volumes API for Amazon S3 and for Google Cloud Storage.

Usage:

```
Imports$bulk_submit_import(items)
```

Arguments:

`items` Nested list of elements containing information about each file/folder to be imported. For each element, users must provide:

- `source_volume` - Volume object or its ID to import files/folders from,
- `source_location` - Volume-specific location pointing to the file or folder to import. This location should be recognizable to the underlying cloud service as a valid key or path to the item. If the item being imported is a folder, its path should end with a `/`.
  Please note that if this volume was configured with a prefix parameter when it was created, the value of prefix will be prepended to the location before attempting to locate the item on the volume.
- `destination_project` - Project object or ID to import files/folders into. Should not be used together with destination_parent. If project is used, the items will be imported to the root of the project's files.
- `destination_parent` - File object of type 'folder' or its ID to import files/folders into. Should not be used together with destination_project. If parent is used, the import will take place into the specified folder, within the project to which the folder belongs.
- `name` - The name of the alias to create. This name should be unique to the project. If the name is already in use in the project, you should use the `autorename` parameter in this call to automatically rename the item (by prefixing its name with an underscore and number).
  If name is omitted, the alias name will default to the last segment of the complete location (including the prefix) on the volume. Segments are considered to be separated with forward slashes ('/').
- `autorename` - Whether to automatically rename the item (by prefixing its name with an underscore and number) if another one with the same name already exists at the destination.
- `preserve_folder_structure` - Whether to keep the exact source folder structure. The default value is TRUE if the item being imported is a folder. Should not be used if you are importing a file.

Example of the list:

```
items <- list(
          list(
            source_volume = 'rfranklin/my-volume',
            source_location = 'chimeras.html.gz',
            destination_project = 'rfranklin/my-project'
          ),
          list(
            source_volume = 'rfranklin/my-volume',
            source_location = 'my-folder/',
            destination_project = 'rfranklin/my-project',
            autorename = TRUE,
            preserve_folder_structure = TRUE
          ),
          list(
            source_volume = 'rfranklin/my-volume',
            source_location = 'my-volume-folder/',
```

```
            destination_parent = '567890abc1e5339df0414123',
            name = 'new-folder-name',
            autorename = TRUE,
            preserve_folder_structure = TRUE
          )
        )
```

Read more on how to import folders from your volume into a project or a project folder. Utility function `prepare_items_for_bulk_import` can help you prepare the `items` parameter based on the provided list of `VolumeFile` or `VolumePrefix` objects.

Returns: `Collection` with list of `Import` objects.

Examples:

```
\dontrun{
 imports_object <- Imports$new(
                    auth = auth
                   )

 # Submit new import into a project
 imports_object$bulk_submit_import(items = list(
   list(
     source_volume = "rfranklin/my-volume",
     source_location = "my-file.txt",
     destination_project = test_project_object,
     autorename = TRUE
   ),
   list(
     source_volume = "rfranklin/my-volume",
     source_location = "my-folder/",
     destination_parent = "parent-folder-id",
     autorename = FALSE,
     preserve_folder_structure = TRUE
   )
  )
 )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Imports$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Imports$query`
## ------------------------------------------------
```

```
## Not run:
 imports_object <- Imports$new(
                    auth = auth
                   )

 # List import job
 imports_object$query()

## End(Not run)


## ------------------------------------------------
## Method `Imports$get`
## ------------------------------------------------

## Not run:
 imports_object <- Imports$new(
                    auth = auth
                   )

 # List import job
 imports_object$get(id = id)

## End(Not run)


## ------------------------------------------------
## Method `Imports$submit_import`
## ------------------------------------------------

## Not run:
 imports_object <- Imports$new(
                    auth = auth
                   )

 # Submit new import into a project
 imports_object$submit_import(
  source_location = volume_file_object,
  destination_project = test_project_object,
  autorename = TRUE
 )

## End(Not run)


## ------------------------------------------------
## Method `Imports$bulk_get`
## ------------------------------------------------

## Not run:
 imports_object <- Imports$new(
                    auth = auth
```

```
                  )

 # List import job
 imports_object$bulk_get(
  imports = list("import-job-id-1", "import-job-id-2")
  )

## End(Not run)


## ------------------------------------------------
## Method `Imports$bulk_submit_import`
## ------------------------------------------------

## Not run:
 imports_object <- Imports$new(
                    auth = auth
                   )

 # Submit new import into a project
 imports_object$bulk_submit_import(items = list(
   list(
     source_volume = "rfranklin/my-volume",
     source_location = "my-file.txt",
     destination_project = test_project_object,
     autorename = TRUE
   ),
   list(
     source_volume = "rfranklin/my-volume",
     source_location = "my-folder/",
     destination_parent = "parent-folder-id",
     autorename = FALSE,
     preserve_folder_structure = TRUE
   )
  )
 )

## End(Not run)
```

`Invoice` *R6 Class representing invoice information.*

Description

R6 Class representing invoice information.

Details

This object contains information about a selected invoice, including costs for analysis, storage, and the invoice period.

Super class

sevenbridges2::Item -> Invoice

Public fields

URL List of URL endpoints for this resource.

id Invoice identifier.

pending Invoice approval status.

approval_date Invoice approval date.

invoice_period Invoicing period (from-to).

analysis_costs Costs of your analysis.

storage_costs Storage costs.

total Total costs.

Methods

Public methods:

- Invoice$new()
- Invoice$print()
- Invoice$reload()
- Invoice$clone()

Method new(): Create new Invoice object.

Usage:

```
Invoice$new(res = NA, ...)
```

Arguments:

res Response containing Invoice object information.

... Other response arguments.

Method print(): Print invoice information as a bullet list.

Usage:

```
Invoice$print()
```

Examples:

```
\dontrun{
 # x is API response when invoice is requested
 invoice_object <- Invoice$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print invoice object
 invoice_object$print()
}
```

Method `reload()`**:** Refresh the Invoice object with updated information.

Usage:

```
Invoice$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Invoice` object.

Examples:

```
\dontrun{
 # x is API response when invoice is requested
 invoice_object <- Invoice$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload invoice object
 invoice_object$reload()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Invoice$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Invoice$print`
## ------------------------------------------------

## Not run:
 # x is API response when invoice is requested
 invoice_object <- Invoice$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print invoice object
 invoice_object$print()

## End(Not run)
```

```
## ------------------------------------------------
## Method `Invoice$reload`
## ------------------------------------------------

## Not run:
 # x is API response when invoice is requested
 invoice_object <- Invoice$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload invoice object
 invoice_object$reload()

## End(Not run)
```

Invoices *R6 Class representing invoices endpoints*

Description

R6 Class representing invoice resource endpoints

Super class

sevenbridges2::Resource -> Invoices

Public fields

URL List of URL endpoints for this resource.

Methods

Public methods:

- Invoices$new()
- Invoices$query()
- Invoices$get()
- Invoices$clone()

Method new(): Create a new Invoices object.

Usage:

```
Invoices$new(...)
```

Arguments:

... Other response arguments.

Method `query()`**:** The call returns information about all your available invoices, unless you use the `billing_group` query parameter to specify the ID of a particular billing group, in which case it will return the invoice incurred by that billing group only.

Usage:

```
Invoices$query(
  billing_group = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`billing_group` ID of a billing group or billing group object you want to list invoices for. Optional.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

... Other arguments that can be passed to core `api()` function like query parameters or 'fields', etc.

Returns: `Collection` of `Invoice` objects.

Examples:

```
\dontrun{
 invoices_object <- Invoices$new(
   auth = auth
 )

 # List all your invoices
 invoices_object$query(billing_group = billing_group)
}
```

Method `get()`**:** This call retrieves information about a selected invoice, including the costs for analysis and storage, and the invoice period. Use the call to list invoices to retrieve the `invoice_id`s for a specified billing group.

Usage:

```
Invoices$get(id, ...)
```

Arguments:

`id` The ID of the invoice you are querying.

... Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Invoice` object.

Examples:

```
\dontrun{
 invoices_object <- Invoices$new(
   auth = auth
 )

 # Get a single invoice by id
 invoices_object$get(id = id)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Invoices$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Invoices$query`
## ------------------------------------------------

## Not run:
 invoices_object <- Invoices$new(
   auth = auth
 )

 # List all your invoices
 invoices_object$query(billing_group = billing_group)

## End(Not run)


## ------------------------------------------------
## Method `Invoices$get`
## ------------------------------------------------

## Not run:
 invoices_object <- Invoices$new(
   auth = auth
 )

 # Get a single invoice by id
 invoices_object$get(id = id)

## End(Not run)
```

Item *R6 Class Representing an Item*

Description

Base class for describing objects: Project, Task, App, File, etc.

Public fields

`response` Raw response from the request.

`href` Item's API request URL.

`auth` Seven Bridges Authentication object.

Methods

Public methods:

- `Item$new()`
- `Item$reload()`
- `Item$clone()`

Method `new()`**:** Create a new Item object.

Usage:

```
Item$new(href = NA, response = NA, auth = NA)
```

Arguments:

`href` Item's API request URL.

`response` Raw API response.

`auth` Seven Bridges Authentication object.

Method `reload()`**:** Reload the Item (resource).

Usage:

```
Item$reload(cls, ...)
```

Arguments:

`cls` Item class object.

`...` Other arguments that can be passed to core `api()` function like 'limit', 'offset', 'fields', etc.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Item$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Member *R6 Class representing a project member*

Description

R6 Class representing a resource for managing project members.

Super class

sevenbridges2::Item -> Member

Public fields

`id` Member's ID.

`username` Member's username.

`email` Member's email.

`type` Member's type.

`permissions` Member's permissions.

Methods

Public methods:

- Member$new()
- Member$print()
- Member$reload()
- Member$clone()

Method `new()`**:** Create a new Member object.

Usage:

```
Member$new(res = NA, ...)
```

Arguments:

`res` Response containing Member object information.

`...` Other response arguments.

Method `print()`**:** Print method for Member class.

Usage:

```
Member$print()
```

Examples:

```
\dontrun{
 # x is API response when member is requested
 member_object <- Member$new(
                    res = x,
                    href = x$href,
```

```
                      auth = auth,
                      response = attr(x, "response")
                     )

 # Print member object
 member_object$print()
}
```

Method `reload()`**:** Reload Member object information.

Usage:

```
Member$reload()
```

Examples:

```
\dontrun{
 # x is API response when member is requested
 member_object <- Member$new(
                      res = x,
                      href = x$href,
                      auth = auth,
                      response = attr(x, "response")
                     )

 # Reload member object
 member_object$reload()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Member$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Member$print`
## ------------------------------------------------

## Not run:
 # x is API response when member is requested
 member_object <- Member$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
```

```
 # Print member object
 member_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Member$reload`
## ------------------------------------------------

## Not run:
 # x is API response when member is requested
 member_object <- Member$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload member object
 member_object$reload()

## End(Not run)
```

`Part` *R6 Class representing a part of the uploading file*

Description

R6 Class representing a resource for managing parts of the files' uploads.

Public fields

`URL` List of URL endpoints for this resource.

`part_number` Part number.

`part_size` Part size.

`url` The URL to which to make the HTTP part upload request.

`expires` ISO 8601 combined date and time representation in Coordinated Universal Time (UTC) by when the HTTP part upload request should be made.

`headers` A map of headers and values that should be set when making the HTTP part upload request.

`success_codes` A list of status codes returned by the HTTP part upload request that should be recognized as success. A successful part upload request should be reported back to the API in a call to report an uploaded file part by passing the information collected from the report object.

`report` Report object.

`etag` ETag received after starting a part upload.

`auth` Authentication object.

`response` Response object.

Methods

Public methods:

- `Part$new()`
- `Part$print()`
- `Part$upload_info_part()`
- `Part$upload_complete_part()`
- `Part$clone()`

Method `new()`**:** Create a new Part object.

Usage:

```
Part$new(
  part_number = NA,
  part_size = NA,
  url = NA,
  expires = NA,
  headers = NA,
  success_codes = NA,
  report = NA,
  etag = NA,
  auth = NA
)
```

Arguments:

`part_number` Part number.

`part_size` Part size.

`url` The URL to which to make the HTTP part upload request.

`expires` Combined date and time representation in UTC by when the HTTP part upload request should be made.

`headers` A map of headers and values that should be set when making the HTTP part upload request.

`success_codes` A list of status codes returned by the HTTP part upload request that should be recognized as success.

`report` Report object.

`etag` ETag received after starting a part upload.

`auth` Seven Bridges Authentication object.

Method `print()`**:** Print method for Part class.

Usage:

```
Part$print()
```

Examples:

```
\dontrun{
 upload_part_object <- Part$new(
                         part_number = part_number,
                         part_size = part_size,
                         auth = auth
                       )

 # Print upload part information
 upload_part_object$print()
}
```

Method `upload_info_part()`**:** Get upload part info.

Usage:

```
Part$upload_info_part(upload_id)
```

Arguments:

`upload_id` Upload object or ID of the upload process to which the part belongs.

Examples:

```
\dontrun{
 upload_part_object <- Part$new(
                         part_number = part_number,
                         part_size = part_size,
                         auth = auth
                       )

 # Get upload part status information
 upload_part_object$upload_info_part(upload_id = upload_id)
}
```

Method `upload_complete_part()`**:** Report an uploaded part.

Usage:

```
Part$upload_complete_part(upload_id)
```

Arguments:

`upload_id` Upload object or ID of the upload process that part belongs to.

Examples:

```
\dontrun{
 upload_part_object <- Part$new(
                         part_number = part_number,
                         part_size = part_size,
                         auth = auth
                       )

 # Report an uploaded part
 upload_part_object$upload_complete_part(upload_id = upload_id)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Part$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Part$print`
## ------------------------------------------------

## Not run:
 upload_part_object <- Part$new(
                        part_number = part_number,
                        part_size = part_size,
                        auth = auth
                       )

 # Print upload part information
 upload_part_object$print()

## End(Not run)

## ------------------------------------------------
## Method `Part$upload_info_part`
## ------------------------------------------------

## Not run:
 upload_part_object <- Part$new(
                        part_number = part_number,
                        part_size = part_size,
                        auth = auth
                       )

 # Get upload part status information
 upload_part_object$upload_info_part(upload_id = upload_id)

## End(Not run)

## ------------------------------------------------
## Method `Part$upload_complete_part`
## ------------------------------------------------

## Not run:
 upload_part_object <- Part$new(
                        part_number = part_number,
                        part_size = part_size,
                        auth = auth
                       )

 # Report an uploaded part
 upload_part_object$upload_complete_part(upload_id = upload_id)
```

```
## End(Not run)
```

`Permission` *R6 Class representing member's permissions*

Description

R6 Class representing member's permissions.

Super class

`sevenbridges2::Item` -> `Permission`

Public fields

`write` Write permission.

`read` Read permission.

`copy` Copy permission.

`execute` Execute permission.

`admin` Admin permission.

Methods

Public methods:

- `Permission$new()`
- `Permission$print()`
- `Permission$reload()`
- `Permission$clone()`

Method `new()`**:** Create a new Permission object.

Usage:

```
Permission$new(
  read = TRUE,
  copy = FALSE,
  write = FALSE,
  execute = FALSE,
  admin = FALSE,
  ...
)
```

Arguments:

`read` User can view file names, metadata, and workflows. They cannot view file contents. All members of a project have read permissions by default. Even if you try setting read permissions to `FALSE`, they will still default to `TRUE`.

`copy` User can view file content, copy, and download files from a project. Set value to `TRUE` to assign the user copy permission. Set to `FALSE` to remove copy permission.

`write` User can add, modify, and remove files and workflows in a project. Set value to `TRUE` to assign the user write permission. Set to `FALSE` to remove write permission.

`execute` User can execute workflows and abort tasks in a project. Set value to `TRUE` to assign the user execute permission. Set to `FALSE` to remove execute permission.

`admin` User can modify another user's permissions on a project, add or remove people from the project and manage funding sources. They also have all of the above permissions. Set value to `TRUE` to assign the user admin permission. Set to `FALSE` to remove admin permission.

`...` Other response arguments.

Method `print()`**:** Print method for Permission class.

Usage:

```
Permission$print()
```

Examples:

```
\dontrun{
 # x is API response when permission is requested
 permission_object <- Permission$new(
                    write = x$write,
                    read = x$read,
                    copy = x$copy,
                    execute = x$execute,
                    admin = x$admin,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
 # Print permission object
 permission_object$print()
}
```

Method `reload()`**:** Reload Permission object information.

Usage:

```
Permission$reload()
```

Examples:

```
\dontrun{
 # x is API response when permission is requested
 permission_object <- Permission$new(
                    write = x$write,
                    read = x$read,
                    copy = x$copy,
                    execute = x$execute,
                    admin = x$admin,
                    href = x$href,
                    auth = auth,
```

```
                      response = attr(x, "response")
                     )

  # Reload permission object
  permission_object$reload()
 }
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Permission$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Permission$print`
## ------------------------------------------------

## Not run:
 # x is API response when permission is requested
 permission_object <- Permission$new(
                    write = x$write,
                    read = x$read,
                    copy = x$copy,
                    execute = x$execute,
                    admin = x$admin,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
 # Print permission object
 permission_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Permission$reload`
## ------------------------------------------------

## Not run:
 # x is API response when permission is requested
 permission_object <- Permission$new(
                    write = x$write,
                    read = x$read,
                    copy = x$copy,
                    execute = x$execute,
                    admin = x$admin,
                    href = x$href,
```

```
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload permission object
 permission_object$reload()

## End(Not run)
```

`prepare_items_for_bulk_export` *Prepare items for bulk export*

Description

Utility function to prepare the `items` parameter, a list of elements containing information about each file to be exported using the `bulk_submit_export()` method.

Usage

```
prepare_items_for_bulk_export(
  files,
  destination_volume,
  destination_location_prefix = NULL,
  overwrite = TRUE,
  properties = NULL
)
```

Arguments

`files`	A list of `File` objects or list of strings (IDs) of the files you are about to export to a volume.
`destination_volume`	Either a `Volume` object or the ID of the volume to which the file will be exported.
`destination_location_prefix`	Character. If the volume has been configured with a prefix parameter, `destination_location_prefix` value will be prepended to location before attempting to create the file on the volume. This parameter can be treated as a path to a new file on the volume. The default value is `NULL`. If you would like to export the file into a folder on the volume, please add folder name as the prefix before the file name in the `"<folder-name>/"` form. Remember to put a slash character ("/") at the end of the string. Keep in mind that the same prefix will be added to all items (files) in the resulting list.

`overwrite` Logical. If this is set to `TRUE` and a named file exists in the project where the alias is about to be created, the existing file will be deleted. `FALSE` by default.

Keep in mind that the same overwrite option will be applied to all items (files) in the resulting list.

`properties` Named list of additional volume properties, like:

- `sse_algorithm` - S3 server-side encryption to use when exporting to this bucket. Supported values: `AES256` (SSE-S3 encryption), `aws:kms`, `null` (no server-side encryption). Default: `AES256`.
- `sse_aws_kms_key_Id`: Applies to type: `s3`. If AWS KMS encryption is used, this should be set to the required KMS key. If not set and `aws:kms` is set as `sse_algorithm`, default KMS key is used.
- `aws_canned_acl`: S3 canned ACL to apply on the object during export. Supported values: any one of S3 canned ACLs; `null` (do not apply canned ACLs). Default: `null`.

Keep in mind that the same properties will be applied to all items (files) in the resulting list.

Details

Based on the provided list of `File` objects or file IDs, this function allows you to set the following fields for each item:

- `source_file`
- `destination_volume`
- `destination_location`
- `overwrite`
- `properties`

However, keep in mind that there are certain constraints:

- The same `destination_volume` applies to all items in the resulting list.
- The same applies to `overwrite` and `properties` parameters.
- By default, the `destination_location` field is populated with the source file name. Upon retrieval of the list of items for bulk export, you can manually update the `destination_location` field for each element of the list as needed. Additionally, you have the flexibility to manually modify any other fields in the list if required.

Value

List of body params items for starting an export job.

See Also

`Exports`, `File`, `Volume`

Examples

```
## Not run:
# Example 1: Prepare 3 items for bulk export action
file_object_1 <- a$files$get(id = "file_1_ID")
file_object_2 <- a$files$get(id = "file_2_ID")
file_object_3 <- a$files$get(id = "file_3_ID")

files_to_export <- list(file_object_1, file_object_2, file_object_3)

prepare_items_for_bulk_export(files_to_export,
  destination_volume = "aws_example_volume"
)

## End(Not run)
## Not run:
# Example 2: Prepare 3 items for bulk export action into some folder
# on the volume - use folder name as prefix before file names
file_object_1 <- a$files$get(id = "file_1_ID")
file_object_2 <- a$files$get(id = "file_2_ID")
file_object_3 <- a$files$get(id = "file_3_ID")

files_to_export <- list(file_object_1, file_object_2, file_object_3)

prepare_items_for_bulk_export(files_to_export,
  destination_volume = "aws_example_volume",
  destination_location_prefix = "example_folder/"
)

## End(Not run)
```

---

prepare_items_for_bulk_import

Prepare items for bulk import

Description

Utility function to prepare the `items` parameter, a list of elements containing information about each file or folder to be imported using the `bulk_submit_import()` method.

Usage

```
prepare_items_for_bulk_import(
  volume_items,
  destination_project = NULL,
  destination_parent = NULL,
  autorename = FALSE,
  preserve_folder_structure = TRUE
)
```

Arguments

volume_items	A list of VolumeFile or VolumePrefix objects to be imported.
destination_project	Destination project ID or Project object. Not required, but either destination_project or destination_parent directory must be provided.
destination_parent	Folder ID or File object (with type = 'FOLDER'). Not required, but either destination_project or destination_parent directory must be provided.
autorename	Logical indicating whether to autorename conflicting files (default is FALSE). Set to TRUE if you want to automatically rename the item (by prefixing its name with an underscore and number) if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder content will be renamed, not the whole folder. Keep in mind that the same autorename option will be applied to all items.
preserve_folder_structure	Logical indicating whether to preserve folder structure. Set to TRUE if you want to keep the exact source folder structure. The default value is TRUE if the item being imported is a folder. Should not be used if you are importing a file. Bear in mind that if you use preserve_folder_structure = FALSE, the response will be the parent folder object containing imported files alongside with other files if they exist. Keep in mind that the same preserve_folder_structure option will be applied to all folders.

Details

Based on the provided list of VolumeFile or VolumePrefix objects, this function allows you to set the following fields for each item:

- source_volume
- source_location
- destination_project or destination_parent
- autorename
- preserve_folder_structure

However, keep in mind that there are certain constraints:

- The same destination_project/destination_parent selection applies to all items in the resulting list.
- The same applies to autorename and preserve_folder_structure parameters.
- This function doesn't allow specification of the name of aliases to create. This is something that should be specified per item, therefore it cannot be applied to the entire list. However, once you have the output of the prepare_items_for_bulk_import function you can manually add the name field to certain items if necessary.

Value

A list of elements containing information about each file/folder to be imported.

See Also

Imports, VolumeFile, VolumePrefix

Examples

```
## Not run:
# Example 1: Prepare 2 items for bulk import action - provide destination
# project
volume_obj_1 <- a$volumes$get
volume_obj_2 <- a$volumes$get

volumes_to_import <- list(volume_obj_1, volume_obj_2)

destination_project <- a$projects$get(id = "project_id")

prepare_items_for_bulk_import(
  volume_items = volumes_to_import,
  destination_project = destination_project
)

## End(Not run)
## Not run:
# Example 2: Prepare 2 items for bulk import action - provide destination
# parent
volume_obj_1 <- a$volumes$get
volume_obj_2 <- a$volumes$get

volumes_to_import <- list(volume_obj_1, volume_obj_2)

destination_parent <- a$files$get(id = "folder_id")

prepare_items_for_bulk_import(
  volume_items = volumes_to_import,
  destination_parent = destination_parent
)

## End(Not run)
```

Project *R6 Class representing a project.*

Description

R6 Class representing a central resource for managing projects.

Super class

sevenbridges2::Item -> Project

Public fields

`URL` List of URL endpoints for this resource.

`id` Project identifier. It consists of project owner's username or if you are using Enterprise, then the Division name and project's short name in form of `<owner_username>/<project-short-name>` or `<division-name>/<project-short-name>`.

`name` Project's name.

`billing_group` The ID of the billing group for the project.

`description` Project's description.

`type` Project's type. All projects have type v2.

`tags` The list of project tags.

`settings` A list which contains detailed project settings. The following fields are part of the settings object:

- `locked` - If set `TRUE`, the project is locked down. Locking down a project prevents any Seven Bridges team member from viewing any information about the task.
- `use_interruptible_instances` - Defines the use of spot instances. If not included in the request, spot instances are enabled by default.
- `use_memoization` - Set to `FALSE` by default. If set to `TRUE` memoization is enabled.
- `use_elastic_disk` - If set to `TRUE` Elastic disk is enabled.
- `intermediate_files` (list) - Contains the following subfields:
  - `retention` - Specifies that intermediate files should be retained for a limited amount of time. The value is always `LIMITED`.
  - `duration` - Specifies intermediate files retention period in hours. The minimum value is `1`. The maximum value is `120` and the default value is `24`.

`root_folder` ID of the project's root folder.

`created_by` Username of the person who created the project.

`created_on` Date and time of project creation.

`modified_on` Date and time describing when the project was last modified.

`permissions` An object containing the information about user's permissions within the project.

`category` Project's category. By default projects are `PRIVATE`.

Methods

Public methods:

- `Project$new()`
- `Project$print()`
- `Project$detailed_print()`
- `Project$reload()`
- `Project$update()`
- `Project$delete()`
- `Project$list_members()`

- Project$add_member()
- Project$remove_member()
- Project$get_member()
- Project$modify_member_permissions()
- Project$list_files()
- Project$create_folder()
- Project$get_root_folder()
- Project$list_apps()
- Project$create_app()
- Project$list_tasks()
- Project$list_imports()
- Project$create_task()
- Project$clone()

Method `new()`**:** Create a new Project object.

Usage:

```
Project$new(res = NA, ...)
```

Arguments:

`res` Response containing Project object information.

`...` Other response arguments.

Method `print()`**:** Basic print method for Project class.

Usage:

```
Project$print()
```

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                     res = x,
                     href = x$href,
                     auth = auth,
                     response = attr(x, "response")
                    )

 # Print project object
 project_object$print()
}
```

Method `detailed_print()`**:** Detailed print method for Project class.

Usage:

```
Project$detailed_print()
```

Details: This method allows users to print all the fields from the Project object more descriptively.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print project object in detail
 project_object$detailed_print()
}
```

Method `reload()`**:** Reload Project object information.

Usage:

```
Project$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload project object
 project_object$reload()
}
```

Method `update()`**:** Method that allows you to edit an already existing project. As a project Admin you can use it to change the `name`, `settings`, `tags` or `billing group` of the project. Users with write permissions in the project can change the project `description`.

Usage:

```
Project$update(
  name = NULL,
  description = NULL,
  billing_group = NULL,
  settings = NULL,
  tags = NULL,
  ...
)
```

Arguments:

`name` New name of the project you are updating.

`description` New description of the project you are updating.

`billing_group` Billing object or ID of a particular billing group you want to set to the project.

`settings` Contains detailed project settings as explained in previous methods. Check our API documentation.

`tags` The list of project tags you want to update.

`...` Other arguments that can be passed to core `api()` function like 'limit', 'offset', 'fields', etc.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Change project object name
 project_object$update(name = name)
}
```

Method `delete()`**:** Method that allows you to delete a project from the platform. It can only be successfully made if you have admin status for the project.
Please be careful when using this method and note that calling it will permanently delete the project from the platform.

Usage:

```
Project$delete()
```

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete project object
 project_object$delete()
}
```

Method `list_members()`**:** Method for listing all the project members.

Usage:

```
Project$list_members(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `Member` objects.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List members in a project
 project_object$list_members()
}
```

Method `add_member()`**:** Method for adding new members to a specified project. The call can only be successfully made by a user who has admin permissions in the project.

Usage:

```
Project$add_member(
  user = NULL,
  email = NULL,
 permissions = list(read = TRUE, copy = FALSE, write = FALSE, execute = FALSE, admin =
    FALSE)
)
```

Arguments:

`user` The Seven Bridges Platform username of the person you want to add to the project or object of class Member containing user's username.

`email` The email address of the person you want to add to the project. This has to be the email address that the person used when registering for an account on the Seven Bridges Platform.

`permissions` List of permissions that will be associated with the project's member. It can contain fields: `read`, `copy`, `write`, `execute` and `admin` with logical fields - `TRUE` if certain

permission is allowed to the user, or FALSE if it's not. Requests to add a project member must include the key permissions. However, if you do not include a value for some permission, it will be set to FALSE by default. The exception to this rule is the read permission, which is the default permission on a project. It enables a user to read project data, including file names, but access file contents.
Example:

```
permissions = list(
  read = TRUE,
  copy = TRUE,
  write = FALSE,
  execute = FALSE,
  admin = FALSE
)
```

Returns: Member object.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add member to a project
 project_object$add_member(
      user = "<username_of_a_user_you_want_to_add>",
      permissions = list(write = TRUE, execute = TRUE)
  )
}
```

Method remove_member(): A method for removing members from the project. It can only be successfully run by a user who has admin privileges in the project.

Usage:

```
Project$remove_member(user)
```

Arguments:

user The Seven Bridges Platform username of the person you want to remove from the project or object of class Member containing user's username.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
```

```
                    response = attr(x, "response")
                   )

 # Remove member from a project
 project_object$remove_member(user = user)
}
```

Method `get_member()`**:** This method returns the information about the member of the specified project.

Usage:

```
Project$get_member(user, ...)
```

Arguments:

`user` The Seven Bridges Platform username of the project member you want to get information about or object of class Member containing user's username.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Member` object.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get member from a project
 project_object$get_member(user = user)
}
```

Method `modify_member_permissions()`**:** This method can be used to edit a user's permissions in a specified project. It can only be successfully made by a user who has admin permissions in the project.

Usage:

```
Project$modify_member_permissions(
  user = NULL,
 permissions = list(read = TRUE, copy = FALSE, write = FALSE, execute = FALSE, admin =
    FALSE)
)
```

Arguments:

`user` The Seven Bridges Platform username of the person you want to modify permissions on the volume for or object of class Member containing user's username.

`permissions` List of permissions that will be associated with the project's member. It can contain fields: `read`, `copy`, `write`, `execute` and `admin` with logical fields - `TRUE` if certain permission is allowed to the user, or `FALSE` if it's not. Requests to add a project member must include the key permissions. However, if you do not include a value for some permission, it will be set to `FALSE` by default. The exception to this rule is the `read` permission, which is the default permission on a project. It enables a user to read project data, including file names, but access file contents.

Example:

```
permissions = list(read = TRUE, copy = TRUE)
```

Returns: `Permission` object.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Modify member permissions in a project
 project_object$modify_member_permissions(
                  user = user,
                  permission = list(read = TRUE, copy = FALSE)
                  )
}
```

Method `list_files()`**:** List all project's files and folders.

Usage:

```
Project$list_files(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `File` objects.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List files in a project
 project_object$list_files()
}
```

Method `create_folder()`**:** Create a new folder under the project's root directory. Every project on the Seven Bridges Platform is represented by a root folder which contains all the files associated with a particular project. You can create first level folders within this root folder by using this function.

Usage:

```
Project$create_folder(name)
```

Arguments:

`name` Folder name.

Returns: `File` object of type 'FOLDER'.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List files in a project
 project_object$create_folder(name = "new_folder")
}
```

Method `get_root_folder()`**:** Get project's root folder object

Usage:

```
Project$get_root_folder()
```

Returns: `File` object of type 'FOLDER'.

Examples:

```
\dontrun{
 # x is API response when project is requested
```

```
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get root folder in a project
 project_object$get_root_folder()
}
```

Method `list_apps()`**:** This call lists all apps in the project.

Usage:

```
Project$list_apps(
  query_terms = NULL,
  id = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`query_terms` A list of search terms used to filter apps based on their details. Each term is case-insensitive and can relate to the app's name, label, toolkit, toolkit version, category, tagline, or description. You can provide a single term (e.g., `list("Compressor")`) or multiple terms (e.g., `list("Expression", "Abundance")`) to search for apps that match all the specified terms. If a term matches any part of the app's details, the app will be included in the results. Search terms can also include phrases (e.g., `list("Abundance estimates input")`), which will search for exact matches within app descriptions or other fields.

`id` Use this parameter to query Project's apps based on their ID.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like other query parameters or 'fields', etc.

Returns: `Collection` of `App` objects.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
```

```
                    )

 # List apps in a project
 project_object$list_apps(query_terms = query_term)
}
```

Method `create_app()`**:** This call creates app in project.

Usage:

```
Project$create_app(
  raw = NULL,
  from_path = NULL,
  name,
  raw_format = c("JSON", "YAML")
)
```

Arguments:

`raw` The body of the request should be a CWL app description saved as a `JSON` or `YAML` file. For a template of this description, try making the call to get raw CWL for an app about an app already in one of your projects. Shouldn't be used together with `from_path` parameter.

`from_path` File containing CWL app description. Shouldn't be used together with raw parameter.

`name` A short name for the app (without any non-alphanumeric characters or spaces).

`raw_format` The type of format used (`JSON` or `YAML`).

Returns: `App` object.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Create app in a project
 project_object$create_app(raw = raw)
}
```

Method `list_tasks()`**:** This call lists all tasks from project you can access.
Read more about how to use query parameters properly here.

Usage:

```
Project$list_tasks(
  status = NULL,
  parent = NULL,
  created_from = NULL,
```

```
  created_to = NULL,
  started_from = NULL,
  started_to = NULL,
  ended_from = NULL,
  ended_to = NULL,
 order_by = c("created_time", "start_time", "name", "end_time", "created_by"),
  order = c("asc", "desc"),
  origin_id = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`status` You can filter the returned tasks by their status. Set the value of status to one of the following values:

- QUEUED
- DRAFT
- RUNNING
- COMPLETED
- ABORTED
- FAILED.

`parent` Provide task ID or task object of the parent task to return all child tasks from that parent. A parent task is a task that specifies the criteria by which to batch its inputs into a series of further sub-tasks, called child tasks. See the documentation on batching tasks for more details on how to run tasks in batches.

`created_from` Enter the starting date string for querying tasks created on the specified date and onwards.

`created_to` Enter the ending date string for querying tasks created until the specified date. You can use it in combination with `created_from` to specify a time interval.

`started_from` Enter the starting date string for querying tasks started on the specified date and onwards.

`started_to` Enter the ending date string for querying tasks started until the specified date.

`ended_from` Enter the starting date string for querying tasks that ended on a specified date.

`ended_to` Enter the ending date string for querying tasks that ended until a specified date.

`order_by` Order returned results by the specified field. Allowed values:
`created_time`, `start_time`, `name`, `end_time` and `created_by`.
Sort can be done only by one column. The default value is `created_time`.

`order` Sort results in ascending or descending order by specifying `asc` or `desc`, respectively. Only taken into account if `order_by` is explicitly specified. The default value is `asc`.

`origin_id` Enter an automation run ID to list all tasks created from the specified automation run.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

... Other arguments that can be passed to core api() function like 'fields', etc.

Returns: Collection of Task objects.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List tasks in a project
 project_object$list_tasks()
}
```

Method `list_imports()`**:** This call lists imports initiated by a particular user into this destination project.

Usage:

```
Project$list_imports(
  volume = NULL,
  state = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`volume` Volume id or Volume object. List all imports from particular volume. Optional.

`state` The state of the import job. Possible values are:

- `PENDING`: the import is queued;
- `RUNNING`: the import is running;
- `COMPLETED`: the import has completed successfully;
- `FAILED`: the import has failed.

Example:

```
state = c("RUNNING", "FAILED")
```

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

... Other arguments that can be passed to core api() function like 'fields', etc.

Returns: Collection of Import objects.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List import jobs in a project
 project_object$list_imports()
}
```

Method `create_task()`**:** This call creates a new task. You can create either a single task or a batch task by using the app's default batching, override batching, or disable batching completely. A parent task is a task that specifies criteria by which to batch its inputs into a series of further subtasks, called child tasks. See the documentation on batching tasks for more details on batching criteria.

Usage:

```
Project$create_task(
  app,
  revision = NULL,
  name = NULL,
  description = NULL,
  execution_settings = NULL,
  inputs = NULL,
  output_location = NULL,
  batch = NULL,
  batch_input = NULL,
  batch_by = NULL,
  use_interruptible_instances = NULL,
  action = NULL,
  ...
)
```

Arguments:

`app` The ID of an app or an App object you want to run. Recall that apps are specified by their projects, in the form `<project_id>/<app_name>`.

`revision` The app revision (version) number.

`name` The name of the task.

`description` An optional description of the task.

`execution_settings` Named list with detailed task execution parameters. Detailed task execution parameters:

- `instance_type`: Possible value is the specific instance type, e.g. `"instance_type" = "c4.2xlarge;ebs-gp2;2000"`;
- `max_parallel_instances`: Maximum number of instances running at the same time. Takes any integer value equal to or greater than 1, e.g. `"max_parallel_instances" = 2.`;

- `use_memoization`: Set to `FALSE` by default. Set to `TRUE` to enable memoization;
- `use_elastic_disk`: Set to `TRUE` to enable Elastic Disk.

Here is an example:

```
execution_settings <- list(
  "instance_type" = "c4.2xlarge;ebs-gp2;2000",
  "max_parallel_instances" = 2,
  "use_memoization" = TRUE,
  "use_elastic_disk" = TRUE
  )
```

`inputs` List of objects. See the section on specifying task inputs for information on creating task input objects. Here is an example with various input types:

```
inputs <- list(
  "input_file"= "<file_id/file_object>",
  "input_directory" = "<folder_id/folder_object>",
  "input_array_string" = list("<string_elem_1>", "<string_elem_2>"),
  "input_boolean" = TRUE,
  "input_double" = 54.6,
  "input_enum" = "enum_1",
  "input_float" = 11.2,
  "input_integer" = "asdf",
  "input_long" = 4212,
  "input_string" = "test_string",
  "input_record" = list(
    "input_record_field_file" = "<file_id/file_object>",
    "input_record_field_integer" = 42
   )
  )
```

`output_location` The output location list allows you to define the exact location where your task outputs will be stored. The location can either be defined for the entire project using the main_location parameter, or individually per each output node, by setting the nodes_override parameter to true and defining individual output node locations within nodes_location. See below for more details.

- `main_location` - Defines the output location for all output nodes in the task. Can be a string path within the project in which the task is created, for example `/Analysis/<task_id>_<task_name>/` or a path on an attached volume, such as `volumes://volume_name/<project_id>/html`. Parts of the path enclosed in angle brackets <> are tokens that are dynamically replaced with corresponding values during task execution.
- `main_location_alias`: The string location (path) in the project that will point to the actual location where the outputs are stored. Used if main_location is defined as a volume path (starting with volumes://), to provide an easy way of accessing output data directly from project files.
- `nodes_override`: Enables defining of output locations for output nodes individually through nodes_location (see below). Set to `TRUE` to be able to define individual locations per output node. Default: `FALSE`. Even if nodes_override is set to `TRUE`, it is not necessary to define output locations for each of the output nodes individually. Data from those output nodes that don't have their locations explicitly defined through nodes_location is

either placed in main_location (if defined) or at the project files root if a main output location is not defined for the task.

- `nodes_location`: List of output paths for individual task output nodes in the following format for each output node:
  ```
  <output-node-id> = list(
  "output_location" = "<output-path>",
  "output_location_alias" = "<alias-path>"
  )
  ```
  Example:
  ```
  b64html = list(
   "output_location" = "volumes://outputs/tasks/mar-19",
   "output_location_alias" = "/rfranklin/tasks/picard"
  )
  ```
  In the example above, b64html is the ID of the output node for which you want to define the output location, while the parameters are defined as follows:
  - `output_location` - Can be a path within the project in which the task is created, for example `/Analysis/<task_id>_<task_name>/` or a path on an attached volume, such as `volumes://volume_name/<project_id>/html`. Also accepts tokens.
  - `output_location_alias` - The location (path) in the project that will point to the exact location where the output is stored. Used if output_location is defined as a volume path (starting with volumes://).

`batch` This is set to `FALSE` by default. Set to `TRUE` to create a batch task and specify the `batch_input` and `batch-by` criteria as described below.

`batch_input` The ID of the input on which you wish to batch. You would typically batch on the input consisting of a list of files. If this parameter is omitted, the default batching criteria defined for the app will be used.

`batch_by` Batching criteria in form of list. For example:

```
batch_by = list(
  type = "CRITERIA",
  criteria = list("metadata.condition")
)
```

`use_interruptible_instances` This field can be `TRUE` or `FALSE`. Set this field to `TRUE` to allow the use of spot instances.

`action` If set to `run`, the task will be run immediately upon creation.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Examples:

```
\dontrun{
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
```

```
                    )

 # Create a task in a project
 project_object$create_task(app = app)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Project$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Project$print`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print project object
 project_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Project$detailed_print`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print project object in detail
 project_object$detailed_print()

## End(Not run)
```

```
## ------------------------------------------------
## Method `Project$reload`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload project object
 project_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `Project$update`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Change project object name
 project_object$update(name = name)

## End(Not run)


## ------------------------------------------------
## Method `Project$delete`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
```

```
 # Delete project object
 project_object$delete()

## End(Not run)


## ------------------------------------------------
## Method `Project$list_members`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List members in a project
 project_object$list_members()

## End(Not run)


## ------------------------------------------------
## Method `Project$add_member`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add member to a project
 project_object$add_member(
      user = "<username_of_a_user_you_want_to_add>",
      permissions = list(write = TRUE, execute = TRUE)
  )

## End(Not run)


## ------------------------------------------------
## Method `Project$remove_member`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
```

```
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Remove member from a project
 project_object$remove_member(user = user)

## End(Not run)


## ------------------------------------------------
## Method `Project$get_member`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get member from a project
 project_object$get_member(user = user)

## End(Not run)


## ------------------------------------------------
## Method `Project$modify_member_permissions`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Modify member permissions in a project
 project_object$modify_member_permissions(
                 user = user,
                 permission = list(read = TRUE, copy = FALSE)
                 )

## End(Not run)
```

```
## ------------------------------------------------
## Method `Project$list_files`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List files in a project
 project_object$list_files()

## End(Not run)


## ------------------------------------------------
## Method `Project$create_folder`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List files in a project
 project_object$create_folder(name = "new_folder")

## End(Not run)


## ------------------------------------------------
## Method `Project$get_root_folder`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get root folder in a project
```

```
 project_object$get_root_folder()

## End(Not run)


## ------------------------------------------------
## Method `Project$list_apps`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List apps in a project
 project_object$list_apps(query_terms = query_term)

## End(Not run)


## ------------------------------------------------
## Method `Project$create_app`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Create app in a project
 project_object$create_app(raw = raw)

## End(Not run)


## ------------------------------------------------
## Method `Project$list_tasks`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
```

```
                    response = attr(x, "response")
                   )

 # List tasks in a project
 project_object$list_tasks()

## End(Not run)


## ------------------------------------------------
## Method `Project$list_imports`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List import jobs in a project
 project_object$list_imports()

## End(Not run)


## ------------------------------------------------
## Method `Project$create_task`
## ------------------------------------------------

## Not run:
 # x is API response when project is requested
 project_object <- Project$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Create a task in a project
 project_object$create_task(app = app)

## End(Not run)
```

`Projects` *R6 Class representing projects endpoints.*

Description

R6 Class representing a Projects resource.

Super class

`sevenbridges2::Resource` -> `Projects`

Public fields

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Projects$new()`
- `Projects$query()`
- `Projects$get()`
- `Projects$delete()`
- `Projects$create()`
- `Projects$clone()`

Method `new()`**:** Create new Projects resource object.

Usage:

```
Projects$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** A method to list all projects available to a particular user. If the username is not provided, all projects available to the currently authenticated user will be listed. Otherwise, projects will be listed for the user whose username is provided. Please keep in mind that this way you will only be able to list projects you are a member of.

More details on how to query projects, you can find in our documentation.

Usage:

```
Projects$query(
  name = NULL,
  owner = NULL,
  tags = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`name` Project's name.

`owner` The username of the owner whose projects you want to query.

`tags` A list of project tags used to filter the query results. Each tag should be provided as a string within the list, and tags may include spaces. For example, both "my_tag_1" and "tag with spaces" are valid tag values. The method will return only projects that have all the specified tags.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like other query parameters or 'fields', etc.

Returns: `Collection` of `Project` objects.

Examples:

```
\dontrun{
 projects_object <- Projects$new(auth = auth)

 # Query projects
 projects_object$query(name = name)
}
```

Method `get()`**:** This call creates a Project object containing the details of a specified project.

Usage:

```
Projects$get(id, ...)
```

Arguments:

`id` Project ID. It consists of project owner's username or if you are using Enterprise, then the Division name and project's short name in form of `<owner_username>/<project-short-name>` or `<division-name>/<project-short-name>`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Project` object.

Examples:

```
\dontrun{
 projects_object <- Projects$new(auth = auth)

 # Get project by id
 projects_object$get(id = id)
}
```

Method `delete()`**:** Method that allows you to delete a project from the platform. It can only be successfully made if you have admin status for the project.
Projects are specified by their IDs, which you can obtain by using `Projects$query()` to list projects or by getting a single project using `Projects$get()`. Please be careful when using this method and note that calling it will permanently delete the project from the platform.

Usage:

```
Projects$delete(project, ...)
```

Arguments:

`project` `Project` object or project ID.

`...` Other arguments that can be passed to core `api()` function as 'fields', etc.

Examples:

```
\dontrun{
 projects_object <- Projects$new(auth = auth)

 # Delete a project
 projects_object$delete(project = project)
}
```

Method `create()`**:** A method for creating a new project.

Usage:

```
Projects$create(
  name,
  billing_group = NULL,
  description = name,
  tags = NULL,
  locked = FALSE,
  controlled = FALSE,
  location = NULL,
  use_interruptible_instances = TRUE,
  use_memoization = FALSE,
  use_elastic_disk = FALSE,
  intermediate_files = list(retention = "LIMITED", duration = 24),
  ...
)
```

Arguments:

`name` The name of the project you are creating.

`billing_group` The Billing object or ID of the billing group for the project.

`description` Description of the project.

`tags` The list of project tags.

`locked` Set this field to `TRUE` to lock down a project. Locking down a project prevents any Seven Bridges team member from viewing any information about the task.

`controlled` Set this field to `TRUE` to define this project as controlled i.e. one which will contain controlled data. Set `FALSE` to define the project as open i.e. one which will contain open data.

`location` Specify the location for this project: `aws:us-east-1` or `aws:us-west-2`.

`use_interruptible_instances` Defines the use of spot instances.

`use_memoization` Set to `FALSE` by default. Set to `TRUE` to enable memoization.

`use_elastic_disk` Set to `TRUE` to enable Elastic disk.

`intermediate_files` A list defining the retention period for intermediate files. Expected elements:

- `retention` - Specifies that intermediate files should be retained for a limited amount of time. The value is always `LIMITED`.
- `duration` - Specifies intermediate files retention period in hours. The minimum value is `1`. The maximum value is `120` and the default value is `24`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Project` object.

Examples:

```
\dontrun{
 projects_object <- Projects$new(auth = auth)

 # Create a project
 projects_object$create(
                    name = name,
                    billing_group = billing_group,
                    description = description
                  )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Projects$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Projects$query`
## ------------------------------------------------

## Not run:
 projects_object <- Projects$new(auth = auth)

 # Query projects
 projects_object$query(name = name)

## End(Not run)


## ------------------------------------------------
## Method `Projects$get`
## ------------------------------------------------

## Not run:
 projects_object <- Projects$new(auth = auth)
```

```
 # Get project by id
 projects_object$get(id = id)

## End(Not run)


## ------------------------------------------------
## Method `Projects$delete`
## ------------------------------------------------

## Not run:
 projects_object <- Projects$new(auth = auth)

 # Delete a project
 projects_object$delete(project = project)

## End(Not run)


## ------------------------------------------------
## Method `Projects$create`
## ------------------------------------------------

## Not run:
 projects_object <- Projects$new(auth = auth)

 # Create a project
 projects_object$create(
                    name = name,
                    billing_group = billing_group,
                    description = description
                  )

## End(Not run)
```

`Rate` *R6 Class Representing a Rate Limit for a user*

Description

Rate object containing information about user's rate limit.

Details

This is the main object for Rate Limit.

Super class

`sevenbridges2::Item` -> `Rate`

Public fields

`rate` A list containing the information about user's current rate limit. It consists of the following fields:

- `limit` Indicates how many requests can be made in five minutes.
- `remaining` Indicates how many requests remain.
- `reset` Indicates the time when the request rate limit will be reset.

`instance` A list containing the information about user's current instance limit. It consists of the following fields:

- `limit` Indicates the total number of instances available to the user. For the first few months, instance limits are unlimited. This is indicated by a special limit of -1. Correspondingly, the remaining value is high.
- `remaining` Indicates the number of the instances that are available at the moment. For the first few months, instance limits are unlimited. This is indicated by a high remaining value. Correspondingly, the limit is set to a special value of -1.

Methods

Public methods:

- `Rate$new()`
- `Rate$print()`
- `Rate$clone()`

Method `new()`**:** Create a new Rate limit object.

Usage:

```
Rate$new(res = NA, ...)
```

Arguments:

`res` Response containing Rate limit object info.

`...` Other response arguments.

Method `print()`**:** Print rate limit information as a bullet list.

Usage:

```
Rate$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Rate$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Resource` *R6 Class Representing a Resource*

Description

Base class for describing a resource.

Details

This is a base class for describing a resource on the platform: Projects, Tasks, Volumes, Files, Apps etc.

Public fields

`auth` Seven Bridges Authentication object.

`URL` List of URL endpoints for this resource.

Methods

Public methods:

- `Resource$new()`
- `Resource$query()`
- `Resource$get()`
- `Resource$delete()`
- `Resource$clone()`

Method `new()`**:** Create a new Resource object.

Usage:

```
Resource$new(auth = NA)
```

Arguments:

`auth` Seven Bridges Authentication object.

Method `query()`**:** Generic query implementation that is used by the resources.

Usage:

```
Resource$query(...)
```

Arguments:

`...` Parameters that will be passed to core `api()` function.

Method `get()`**:** Generic get implementation that fetches a single resource from the server.

Usage:

```
Resource$get(cls, id, ...)
```

Arguments:

`cls` Resource class object.

`id` Object id.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Method `delete()`**:** Generic implementation to delete a resource from the server.

Usage:

```
Resource$delete(id, ...)
```

Arguments:

`id` Object id.

`...` Other arguments that can be passed to core `api()` function.

`cls` Resource class object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Resource$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Task` *R6 Class representing a Task*

Description

R6 Class representing a resource for managing tasks.

Super class

`sevenbridges2::Item` -> `Task`

Public fields

`URL` List of URL endpoints for this resource.

`id` The ID of the task.

`name` The name of the task.

`status` Task status (different from execution_status). Allowed values:

- QUEUED
- DRAFT
- RUNNING
- COMPLETED
- ABORTED
- FAILED

`description` An optional description of a task.

`project` Identifier of the project that the task is located in.

`app` The identifier of the app that was used for the task.

`created_by` Username of the task creator.

`executed_by` Username of the task executor.

`created_on` The time in form of string when the task was created.

`start_time` Task start time in form of string.

`end_time` Task end time in form of string.

`origin` ID of the entity that created the task, e.g., an automation run, if task was created by an automation run.

`use_interruptible_instances` This field can be `TRUE` or `FALSE`. Set this field to `TRUE` to allow the use of spot instances.

`batch` `TRUE` for batch tasks, `FALSE` for regular and child tasks (batch this task; if `FALSE`, will not create a batch task).

`batch_by` Batching criteria (list).

`batch_group` Batch group for a batch task (list). Represents the group that is assigned to the child task from the batching criteria that was used when the task was started.

`batch_input` Input identifier on to which to apply batching.

`batch_parent` Parent task ID for a batch child. (batch task which is the parent of this task).

`execution_settings` Execution settings list for the task.

`execution_status` Task execution status list - info about current execution status.

`errors` Validation errors list stored as a high-level errors array property in the API response.

`warnings` Validation warnings list from API response.

`price` Task cost (list) - contains amount and currency.

`inputs` List of inputs that were submitted to the task.

`outputs` List of generated outputs from the task.

`output_location` List of locations where task outputs will be stored.

Methods

Public methods:

- `Task$new()`
- `Task$print()`
- `Task$reload()`
- `Task$run()`
- `Task$abort()`
- `Task$clone_task()`
- `Task$get_execution_details()`
- `Task$list_batch_children()`
- `Task$delete()`
- `Task$rerun()`
- `Task$update()`
- `Task$clone()`

Method `new()`**:** Create new Task object.

Usage:

```
Task$new(res = NA, ...)
```

Arguments:

`res` Response containing Task object information.

`...` Other response arguments.

Method `print()`**:** Print method for Task class.

Usage:

```
Task$print()
```

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print task object
 task_object$print()
}
```

Method `reload()`**:** Reload Task object information.

Usage:

```
Task$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload task object
 task_object$reload()
}
```

Method `run()`: This call runs (executes) the task. Only tasks whose status is `DRAFT` can be run.

Usage:

```
Task$run(
  batch = NULL,
  use_interruptible_instances = NULL,
  in_place = TRUE,
  ...
)
```

Arguments:

`batch` Set this to `FALSE` to disable the default batching for this task. Running a batch task is a recommended way to run multiple tasks considering the API rate limit (learn more).

`use_interruptible_instances` This field can be `TRUE` or `FALSE`. Set this field to `TRUE` to allow the use of spot instances.

`in_place` Default `TRUE`. Should the new object of Task class be returned or the current to be reinitialized.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Run task
 task_object$run()
}
```

Method `abort()`: This call aborts the specified task. Only tasks whose status is `RUNNING` or `QUEUED` may be aborted.

Usage:

```
Task$abort(in_place = TRUE, ...)
```

Arguments:

`in_place` Default `TRUE`. Should the new object of Task class be returned or the current to be reinitialized.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Examples:

```
\dontrun{
# x is API response when task is requested
```

```
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

. # Run task
 task_object$run()

 # Then abort task
 task_object$abort()
}
```

Method `clone_task()`**:** This call clones the specified task. Once cloned, the task can either be in `DRAFT` mode or immediately ran, by setting the `run` parameter to `TRUE`.

Usage:

```
Task$clone_task(run = FALSE, ...)
```

Arguments:

`run` Set this to `TRUE` in order to create a draft task and execute it immediately. Default: `FALSE`.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Clone task object
 task_object$clone_task()
}
```

Method `get_execution_details()`**:** This call returns execution details of the specified task. The task is referred to by its ID, which you can obtain by making the call to list all tasks you can access. The call breaks down the information into the task's distinct jobs. A job is a single subprocess carried out in a task. The information returned by this call is broadly similar to that which can be found in the task stats and logs provided on the Platform. The task execution details include the following information:

- The name of the command line job that executed
- The start time of the job

- End time of the job (if it completed)
- The status of the job (DONE, FAILED, or RUNNING)
- Information on the computational instance that the job was run on, including the provider ID, the type of instance used and the cloud service provider
- A link that can be used to download the standard error logs for the job.
- SHA hash of the Docker image ('checksum').

Usage:

```
Task$get_execution_details(...)
```

Arguments:

... Other arguments that can be passed to core api() function like 'fields', etc.

Returns: List of execution details.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get task execution details
 task_object$get_execution_details()
}
```

Method `list_batch_children()`**:** This call retrieves batch child tasks for this task if its a batch task.

Usage:

```
Task$list_batch_children(
  status = NULL,
  project = NULL,
  created_from = NULL,
  created_to = NULL,
  started_from = NULL,
  started_to = NULL,
  ended_from = NULL,
  ended_to = NULL,
  order_by = NULL,
  order = NULL,
  origin_id = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`status` You can filter the returned tasks by their status. Set the value of status to one of the following values:

- QUEUED
- DRAFT
- RUNNING
- COMPLETED
- ABORTED
- FAILED.

`project` Provide the project ID or Project object you wish to list the tasks from.

`created_from` Enter the starting date string for querying tasks created on the specified date and onwards.

`created_to` Enter the ending date string for querying tasks created until the specified date. You can use it in combination with `created_from` to specify a time interval.

`started_from` Enter the starting date string for querying tasks started on the specified date and onwards.

`started_to` Enter the ending date string for querying tasks started until the specified date.

`ended_from` Enter the starting date string for querying tasks that ended on a specified date.

`ended_to` Enter the ending date string for querying tasks that ended until a specified date.

`order_by` Order returned results by the specified field. Allowed values: `created_time`, `start_time`, `name`, `end_time` and `created_by`. Sort can be done only by one column. The default value is `created_time`.

`order` Sort results in ascending or descending order by specifying `asc` or `desc`, respectively. Only taken into account if `order_by` is explicitly specified. The default value is `asc`.

`origin_id` Enter an automation run ID to list all tasks created from the specified automation run.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `Task` objects.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List batch children of a task
```

```
 task_object$list_batch_children()
}
```

Method `delete()`**:** This call deletes the specified task. The task is referred to by its ID, which you can obtain by making the call to list all tasks you can access.

Usage:

```
Task$delete(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete task
 task_object$delete()
}
```

Method `rerun()`**:** This call reruns (executes) the specified task.

Usage:

```
Task$rerun(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Rerun task
 task_object$rerun()
}
```

Method `update()`**:** Change the details of the specified task, including its name, description, and inputs. Note that you can only modify tasks with a task status of `DRAFT`. Tasks which are `RUNNING`, `QUEUED`, `ABORTED`, `COMPLETED` or `FAILED` cannot be modified in order to enable the reproducibility of analyses which have been queued for execution or has initiated executing. There are two things to note if you are editing a batch task:

- 1 If you want to change the input on which to batch and the batch criteria, you need to specify the batch_input and batch_by parameters together in the same function call.
- 2 If you want to disable batching on a task, set batch to false. Or, you can also set the parameters batch_input and batch_by to NULL.

Usage:

```
Task$update(
  name = NULL,
  description = NULL,
  execution_settings = NULL,
  inputs = NULL,
  output_location = NULL,
  batch = NULL,
  batch_input = NULL,
  batch_by = NULL,
  ...
)
```

Arguments:

`name` The name of the task.

`description` An optional description of the task.

`execution_settings` Named list with detailed task execution parameters. Detailed task execution parameters:

- `instance_type`: Possible value is the specific instance type, e.g. `"instance_type" = "c4.2xlarge;ebs-gp2;2000"`;
- `max_parallel_instances`: Maximum number of instances running at the same time. Takes any integer value equal to or greater than 1, e.g. `"max_parallel_instances" = 2.`;
- `use_memoization`: Set to `FALSE` by default. Set to `TRUE` to enable memoization;
- `use_elastic_disk`: Set to `TRUE` to enable Elastic Disk.

Here is an example:

```
execution_settings <- list(
  "instance_type" = "c4.2xlarge;ebs-gp2;2000",
  "max_parallel_instances" = 2,
  "use_memoization" = TRUE,
  "use_elastic_disk" = TRUE
  )
```

`inputs` List of objects. See the section on specifying task inputs for information on creating task input objects. Here is an example with various input types:

```
inputs <- list(
  "input_file"= "<file_id/file_object>",
  "input_directory" = "<folder_id/folder_object>",
```

```
"input_array_string" = list("<string_elem_1>", "<string_elem_2>"),
"input_boolean" = TRUE,
"input_double" = 54.6,
"input_enum" = "enum_1",
"input_float" = 11.2,
"input_integer" = "asdf",
"input_long" = 4212,
"input_string" = "test_string",
"input_record" = list(
  "input_record_field_file" = "<file_id/file_object>",
  "input_record_field_integer" = 42
 )
)
```

`output_location` The output location list allows you to define the exact location where your task outputs will be stored. The location can either be defined for the entire project using the main_location parameter, or individually per each output node, by setting the nodes_override parameter to true and defining individual output node locations within nodes_location. See below for more details.

- `main_location` - Defines the output location for all output nodes in the task. Can be a string path within the project in which the task is created, for example `/Analysis/<task_id>_<task_name>/` or a path on an attached volume, such as `volumes://volume_name/<project_id>/html`. Parts of the path enclosed in angle brackets <> are tokens that are dynamically replaced with corresponding values during task execution.
- `main_location_alias`: The string location (path) in the project that will point to the actual location where the outputs are stored. Used if main_location is defined as a volume path (starting with volumes://), to provide an easy way of accessing output data directly from project files.
- `nodes_override`: Enables defining of output locations for output nodes individually through nodes_location (see below). Set to `TRUE` to be able to define individual locations per output node. Default: `FALSE`. Even if nodes_override is set to `TRUE`, it is not necessary to define output locations for each of the output nodes individually. Data from those output nodes that don't have their locations explicitly defined through nodes_location is either placed in main_location (if defined) or at the project files root if a main output location is not defined for the task.
- `nodes_location`: List of output paths for individual task output nodes in the following format for each output node:

  ```
  <output-node-id> = list(
  "output_location" = "<output-path>",
  "output_location_alias" = "<alias-path>"
  )
  ```

  Example:

  ```
  b64html = list(
   "output_location" = "volumes://outputs/tasks/mar-19",
   "output_location_alias" = "/rfranklin/tasks/picard"
  )
  ```

  In the example above, b64html is the ID of the output node for which you want to define the output location, while the parameters

are defined as follows:

- output_location - Can be a path within the project in which the task is created, for example /Analysis/<task_id>_<task_name>/ or a path on an attached volume, such as volumes://volume_name/<project_id>/html. Also accepts tokens.
- output_location_alias - The location (path) in the project that will point to the exact location where the output is stored. Used if output_location is defined as a volume path (starting with volumes://).

batch This is set to FALSE by default. Set to TRUE to create a batch task and specify the batch_input and batch-by criteria as described below.

batch_input The ID of the input on which you wish to batch. You would typically batch on the input consisting of a list of files. If this parameter is omitted, the default batching criteria defined for the app will be used.

batch_by Batching criteria in form of list. For example:

```
batch_by = list(
  type = "CRITERIA",
  criteria = list("metadata.condition")
)
```

... Other arguments that can be passed to core api() function like 'fields', etc.

Returns: Task object.

Examples:

```
\dontrun{
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Update task
 task_object$update(name = new_name)
}
```

Method clone()**:** The objects of this class are cloneable with this method.

Usage:

```
Task$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Task$print`
## ------------------------------------------------
```

```
## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print task object
 task_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Task$reload`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload task object
 task_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `Task$run`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Run task
 task_object$run()

## End(Not run)
```

```
## ------------------------------------------------
## Method `Task$abort`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

. # Run task
 task_object$run()

 # Then abort task
 task_object$abort()

## End(Not run)


## ------------------------------------------------
## Method `Task$clone_task`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Clone task object
 task_object$clone_task()

## End(Not run)


## ------------------------------------------------
## Method `Task$get_execution_details`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )
```

```
 # Get task execution details
 task_object$get_execution_details()

## End(Not run)


## ------------------------------------------------
## Method `Task$list_batch_children`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List batch children of a task
 task_object$list_batch_children()

## End(Not run)


## ------------------------------------------------
## Method `Task$delete`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete task
 task_object$delete()

## End(Not run)


## ------------------------------------------------
## Method `Task$rerun`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
```

```
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Rerun task
 task_object$rerun()

## End(Not run)


## ------------------------------------------------
## Method `Task$update`
## ------------------------------------------------

## Not run:
# x is API response when task is requested
task_object <- Task$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Update task
 task_object$update(name = new_name)

## End(Not run)
```

Tasks *R6 Class representing tasks endpoints*

Description

R6 Class representing tasks resource endpoints.

Super class

sevenbridges2::Resource -> Tasks

Public fields

URL List of URL endpoints for this resource.

Methods

Public methods:

- Tasks$new()

- Tasks$query()
- Tasks$get()
- Tasks$delete()
- Tasks$create()
- Tasks$bulk_get()
- Tasks$clone()

Method `new()`**:** Create new Tasks resource object.

Usage:

```
Tasks$new(...)
```

Arguments:

`...` Other response arguments.

Method `query()`**:** This call lists all tasks you can access.

Read more about how to use query parameters properly here.

Usage:

```
Tasks$query(
  status = NULL,
  parent = NULL,
  project = NULL,
  created_from = NULL,
  created_to = NULL,
  started_from = NULL,
  started_to = NULL,
  ended_from = NULL,
  ended_to = NULL,
 order_by = c("created_time", "start_time", "name", "end_time", "created_by"),
  order = c("asc", "desc"),
  origin_id = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`status` You can filter the returned tasks by their status. Set the value of status to one of the following values: `QUEUED`, `DRAFT`, `RUNNING`, `COMPLETED`, `ABORTED`, `FAILED`.

`parent` Provide the task ID or Task object of the parent task to return all child tasks. A parent task is a task that specifies criteria by which to batch its inputs into a series of further subtasks, called child tasks. See the documentation on batching tasks for more details on how to run tasks in batches.

`project` Provide the project ID or Project object you wish to list the tasks from.

`created_from` Enter the starting date string for querying tasks created on the specified date and onwards.

`created_to` Enter the ending date string for querying tasks created until the specified date. You can use it in combination with `created_from` to specify a time interval.

`started_from` Enter the starting date string for querying tasks started on the specified date and onwards.

`started_to` Enter the ending date string for querying tasks started until the specified date.

`ended_from` Enter the starting date string for querying tasks that ended on a specified date.

`ended_to` Enter the ending date string for querying tasks that ended until a specified date.

`order_by` Order returned results by the specified field. Allowed values: `created_time`, `start_time`, `name`, `end_time` and `created_by`. Sorting can only be done by one column. The default value is `created_time`.

`order` Sort results in ascending or descending order by specifying `asc` or `desc`, respectively. Only taken into account if `order_by` is explicitly specified. The default value is `asc`.

`origin_id` Enter an automation run ID to list all tasks created from the specified automation run.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` of `Task` objects.

Method `get()`**:** This call returns details of the specified task. The task is referred to by its ID, which you can obtain by making the call to list all tasks you can access. The task details include its creator, its start and end time, the number of jobs completed in it, and its input and output files. You can also see the status of the task.

Usage:

```
Tasks$get(id, ...)
```

Arguments:

`id` The ID of the task you are querying.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Method `delete()`**:** Tasks are identified by their IDs, which can be obtained using `Tasks$query()` to list tasks or `Tasks$get()` to retrieve a single task.

Usage:

```
Tasks$delete(task, ...)
```

Arguments:

`task` `Task` object or task ID.

`...` Other arguments that can be passed to core `api()` function as 'fields', etc.

Method `create()`**:** This call creates a new task. You can create either a single task or a batch task by using the app's default batching, override batching, or disable batching completely. A parent task is a task that specifies criteria by which to batch its inputs into a series of further sub-tasks, called child tasks. the documentation on batching tasks for more details on batching criteria.

Usage:

```
Tasks$create(
  project,
  app,
  revision = NULL,
  name = NULL,
  description = NULL,
  execution_settings = NULL,
  inputs = NULL,
  output_location = NULL,
  batch = NULL,
  batch_input = NULL,
  batch_by = NULL,
  use_interruptible_instances = NULL,
  action = NULL,
  ...
)
```

Arguments:

`project` The ID of a project or a Project object where you want to create the task in.

`app` The ID of an app or an App object you want to run. Recall that apps are specified by their projects, in the form `<project_id>/<app_name>`.

`revision` The app revision (version) number.

`name` The name of the task.

`description` An optional description of the task.

`execution_settings` Named list with detailed task execution parameters. Detailed task execution parameters:

- `instance_type`: Possible value is the specific instance type, e.g. `"instance_type" = "c4.2xlarge;ebs-gp2;2000"`;
- `max_parallel_instances`: Maximum number of instances running at the same time. Takes any integer value equal to or greater than 1, e.g. `"max_parallel_instances" = 2.`;
- `use_memoization`: Set to `FALSE` by default. Set to `TRUE` to enable memoization;
- `use_elastic_disk`: Set to `TRUE` to enable Elastic Disk.

Here is an example:

```
execution_settings <- list(
  "instance_type" = "c4.2xlarge;ebs-gp2;2000",
  "max_parallel_instances" = 2,
  "use_memoization" = TRUE,
  "use_elastic_disk" = TRUE
  )
```

`inputs` List of objects. See the section on specifying task inputs for information on creating task input objects. Here is an example with various input types:

```
inputs <- list(
  "input_file"= "<file_id/file_object>",
  "input_directory" = "<folder_id/folder_object>",
  "input_array_string" = list("<string_elem_1>", "<string_elem_2>"),
```

```
"input_boolean" = TRUE,
"input_double" = 54.6,
"input_enum" = "enum_1",
"input_float" = 11.2,
"input_integer" = "asdf",
"input_long" = 4212,
"input_string" = "test_string",
"input_record" = list(
  "input_record_field_file" = "<file_id/file_object>",
  "input_record_field_integer" = 42
 )
)
```

`output_location` The output location list allows you to define the exact location where your task outputs will be stored. The location can either be defined for the entire project using the main_location parameter, or individually per each output node, by setting the nodes_override parameter to true and defining individual output node locations within nodes_location. See below for more details.

- `main_location` - Defines the output location for all output nodes in the task. Can be a string path within the project in which the task is created, for example
  `/Analysis/<task_id>_<task_name>/`
  or a path on an attached volume, such as
  `volumes://volume_name/<project_id>/html`.
  Parts of the path enclosed in angle brackets <> are tokens that are dynamically replaced with corresponding values during task execution.
- `main_location_alias`: The string location (path) in the project that will point to the actual location where the outputs are stored. Used if main_location is defined as a volume path (starting with volumes://), to provide an easy way of accessing output data directly from project files.
- `nodes_override`: Enables defining of output locations for output nodes individually through nodes_location (see below). Set to `TRUE` to be able to define individual locations per output node. Default: `FALSE`. Even if nodes_override is set to `TRUE`, it is not necessary to define output locations for each of the output nodes individually. Data from those output nodes that don't have their locations explicitly defined through nodes_location is either placed in main_location (if defined) or at the project files root if a main output location is not defined for the task.
- `nodes_location`: List of output paths for individual task output nodes in the following format for each output node:
  ```
  <output-node-id> = list(
  "output_location" = "<output-path>",
  "output_location_alias" = "<alias-path>"
  )
  ```
  Example:
  ```
  b64html = list(
   "output_location" = "volumes://outputs/tasks/mar-19",
   "output_location_alias" = "/rfranklin/tasks/picard"
  )
   In the example above, b64html is the ID of the output node for
   which you want to define the output location, while the parameters
  ```

are defined as follows:

- `output_location` - Can be a path within the project in which the task is created, for example `/Analysis/<task_id>_<task_name>/` or a path on an attached volume, such as `volumes://volume_name/<project_id>/html`. Also accepts tokens.
- `output_location_alias` - The location (path) in the project that will point to the exact location where the output is stored. Used if output_location is defined as a volume path (starting with volumes://).

`batch` This is set to `FALSE` by default. Set to `TRUE` to create a batch task and specify the `batch_input` and `batch-by` criteria as described below.

`batch_input` The ID of the input on which you wish to batch. You would typically batch on the input consisting of a list of files. If this parameter is omitted, the default batching criteria defined for the app will be used.

`batch_by` Batching criteria in form of list. For example:

```
batch_by = list(
  type = "CRITERIA",
  criteria = list("metadata.condition")
)
```

`use_interruptible_instances` This field can be `TRUE` or `FALSE`. Set this field to `TRUE` to allow the use of spot instances.

`action` If set to `run`, the task will be run immediately upon creation.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Task` object.

Method `bulk_get()`**:** This call returns statistics for all specified tasks.

Usage:

```
Tasks$bulk_get(tasks)
```

Arguments:

`tasks` A list of `Task` objects or list of strings (IDs) of the tasks you are requesting the statistics for.

Returns: `Collection` (list of `Task` objects).

Examples:

```
\dontrun{
 # Get details of multiple tasks
 a$tasks$bulk_get(
               tasks = list("task_1_ID", "task_2_ID")
              )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Tasks$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Tasks$bulk_get`
## ------------------------------------------------

## Not run:
 # Get details of multiple tasks
 a$tasks$bulk_get(
               tasks = list("task_1_ID", "task_2_ID")
              )

## End(Not run)
```

Team *R6 Class representing a Team*

Description

R6 Class representing a central resource for managing teams.

Super class

sevenbridges2::Item -> Team

Public fields

URL List of URL endpoints for this resource.

id The ID of the team.

name Team's name.

Methods

Public methods:

- Team$new()
- Team$print()
- Team$reload()
- Team$list_members()
- Team$add_member()
- Team$remove_member()
- Team$rename()
- Team$delete()
- Team$clone()

Method new(): Create a new Team object.

Usage:

```
Team$new(res = NA, ...)
```

Arguments:

`res` Response containing the Team object information.

`...` Other response arguments.

Method `print()`**:** Print method for Team class.

Usage:

```
Team$print()
```

Examples:

```
\dontrun{
 team_object <- Team$new(
  res = x,
href = x$href,
auth = auth,
response = attr(x, "response")
 )
 team_object$print()
}
```

Method `reload()`**:** Reload Team object information.

Usage:

```
Team$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Team` object.

Examples:

```
\dontrun{
 team_object <- Team$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 team_object$reload()
}
```

Method `list_members()`**:** This call retrieves a list of all team members within a specified team. Each member's username will be returned.

Usage:

```
Team$list_members(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: A `Collection` of `User` objects.

Examples:

```
\dontrun{
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Retrieve a list of all team members
  my_team$list_members()
}
```

Method `add_member()`**:** This call adds a division member to the specified team. This action requires `ADMIN` privileges.

Usage:

```
Team$add_member(user)
```

Arguments:

`user` User ID of the division member you are adding to the team using the following format: `division_id/username`. Alternatively, a `User` object can be provided.

Examples:

```
\dontrun{
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Add new member to the team
  my_team$add_member(user = "user-id")
}
```

Method `remove_member()`**:** This call removes a member from a team. By removing a member, you remove the user's membership to the team, but do not remove their account from the division. This action requires `ADMIN` privileges.

Usage:

```
Team$remove_member(user)
```

Arguments:

`user` The Seven Bridges Platform username of the user to be removed, specified in the format `division-name/username`, or an object of class `User` that contains the username.

Examples:

```
\dontrun{
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Remove a member from the team
  my_team$remove_member(user = "user-id")
}
```

Method `rename()`**:** This call renames the specified team. This action requires `ADMIN` privileges.

Usage:

```
Team$rename(name = NULL)
```

Arguments:

`name` The new name for the team.

Examples:

```
\dontrun{
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Rename the team
  my_team$rename(name = "new-team-name")
}
```

Method `delete()`**:** This call deletes a team. By deleting a team, you remove the users' membership to the team, but do not remove their accounts from the division. This action requires `ADMIN` privileges.

Usage:

```
Team$delete()
```

Examples:

```
\dontrun{
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Delete a team
  my_team$delete()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Team$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Team$print`
## ------------------------------------------------

## Not run:
 team_object <- Team$new(
  res = x,
href = x$href,
auth = auth,
response = attr(x, "response")
 )
 team_object$print()

## End(Not run)

## ------------------------------------------------
## Method `Team$reload`
## ------------------------------------------------

## Not run:
 team_object <- Team$new(
   res = x,
   href = x$href,
   auth = auth,
   response = attr(x, "response")
 )
 team_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `Team$list_members`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Retrieve a list of all team members
  my_team$list_members()

## End(Not run)


## ------------------------------------------------
## Method `Team$add_member`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified team
```

```
  my_team <- a$teams$get(id = "team-id")

  # Add new member to the team
  my_team$add_member(user = "user-id")

## End(Not run)

## ------------------------------------------------
## Method `Team$remove_member`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Remove a member from the team
  my_team$remove_member(user = "user-id")

## End(Not run)

## ------------------------------------------------
## Method `Team$rename`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Rename the team
  my_team$rename(name = "new-team-name")

## End(Not run)

## ------------------------------------------------
## Method `Team$delete`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified team
  my_team <- a$teams$get(id = "team-id")

  # Delete a team
  my_team$delete()

## End(Not run)
```

`Teams` *R6 Class representing teams endpoints*

Description

R6 Class representing teams resource endpoints.

Super class

sevenbridges2::Resource -> Teams

Public fields

URL List of URL endpoints for this resource.

Methods

Public methods:

- Teams$new()
- Teams$query()
- Teams$get()
- Teams$create()
- Teams$delete()
- Teams$clone()

Method new(): Create new Teams resource object.

Usage:

```
Teams$new(...)
```

Arguments:

... Other response arguments.

Method query(): This call retrieves a list of all teams in a division that you are a member of. Each team's ID and name will be returned.

Usage:

```
Teams$query(division, list_all = FALSE, ...)
```

Arguments:

division The string ID of the division or Division object you are querying.

list_all Boolean. Set this field to TRUE if you want to list all teams within the division (regardless of whether you are a member of a team or not). Default value is FALSE.

... Other arguments that can be passed to core api() function like 'fields', etc.

Returns: A Collection of Team objects.

Examples:

```
\dontrun{
  # Retrieve a list of all teams within the division regardless of
  # whether you are a member of a team or not
  a$teams$query(division = "division-id", list_all = TRUE)
}
```

Method get(): This call returns the details of a specified team. You can only get details of a team you are a member of.

Usage:

```
Teams$get(id, ...)
```

Arguments:

`id` The ID of the team you are querying. The function also accepts a Team object and extracts the ID.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Team` object.

Examples:

```
\dontrun{
  # Retrieve details of a specified team
  a$teams$get(id = "team-id")
}
```

Method `create()`**:** This call creates a new team within a specified division.

Usage:

```
Teams$create(division, name)
```

Arguments:

`division` The string ID of the division or Division object where you want to create a team.

`name` Enter the name for the new team.

Returns: A `Team` object.

Examples:

```
\dontrun{
  # Create new team
  a$teams$create(division = "division-id", name = "my-new-team")
}
```

Method `delete()`**:** This call deletes a team. By deleting a team, you remove the users' membership to the team, but do not remove their accounts from the division.

Usage:

```
Teams$delete(team, ...)
```

Arguments:

`team` The team ID or Team object that you want to delete.

`...` Other arguments that can be passed to core `api()` function.

Examples:

```
\dontrun{
  # Delete a team
  a$teams$delete(team = "team-id")
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Teams$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Teams$query`
## ------------------------------------------------

## Not run:
  # Retrieve a list of all teams within the division regardless of
  # whether you are a member of a team or not
  a$teams$query(division = "division-id", list_all = TRUE)

## End(Not run)


## ------------------------------------------------
## Method `Teams$get`
## ------------------------------------------------

## Not run:
  # Retrieve details of a specified team
  a$teams$get(id = "team-id")

## End(Not run)

## ------------------------------------------------
## Method `Teams$create`
## ------------------------------------------------

## Not run:
  # Create new team
  a$teams$create(division = "division-id", name = "my-new-team")

## End(Not run)


## ------------------------------------------------
## Method `Teams$delete`
## ------------------------------------------------

## Not run:
  # Delete a team
  a$teams$delete(team = "team-id")

## End(Not run)
```

Upload *R6 Class representing an Upload job*

Description

R6 Class representing a resource for managing files' uploads.

Public fields

`URL` List of URL endpoints for this resource.

`upload_id` Upload ID received after upload initialization.

`path` Relative or absolute path to the file on the local disk.

`project` Project's identifier (character).

`parent` The ID of the folder to which the item is being uploaded. Should not be used together with 'project'.

`filename` File name. By default it will be the same as the name of the file you want to upload. However, it can be changed to new name.

`overwrite` If `TRUE` will overwrite file on the server.

`file_size` File size.

`part_size` Size of part in bytes.

`part_length` Number of parts to upload.

`parts` List of parts to be uploaded (class `Part`).

`initialized` If `TRUE`, upload has been initialized.

`auth` Authentication object.

Methods

Public methods:

- `Upload$new()`
- `Upload$print()`
- `Upload$init()`
- `Upload$info()`
- `Upload$start()`
- `Upload$abort()`
- `Upload$clone()`

Method `new()`**:** Create a new Upload object.

Usage:

```
Upload$new(
  path = NA,
  project = NA,
  parent = NA,
  filename = NA,
  overwrite = FALSE,
  file_size = NA,
  part_size = getOption("sevenbridges2")$RECOMMENDED_PART_SIZE,
  initialized = FALSE,
  auth = NA
)
```

Arguments:

`path` Path to the file on the local disc.

`project` Project's identifier (character).

`parent` The ID of the folder to which the item is being uploaded.

`filename` New file name. Optional.

`overwrite` If true will overwrite file on the server.

`file_size` File size.

`part_size` Size of a single part in bytes.

`initialized` If `TRUE`, upload has been initialized.

`auth` Seven Bridges Authentication object.

Method `print()`**:** Print method for Upload class.

Usage:

```
Upload$print()
```

Examples:

```
\dontrun{
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Print the upload object information
 upload_object$print(name = name)
}
```

Method `init()`**:** Initialize a new multipart file upload.

Usage:

```
Upload$init()
```

Examples:

```
\dontrun{
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Initialize multipart file upload object
 upload_object$init()
}
```

Method `info()`**:** Get the details of an active multipart upload.

Usage:

```
Upload$info(list_parts = FALSE)
```

Arguments:

`list_parts` If `TRUE`, also return a list of parts that have been reported as completed for this multipart upload. Please bear in mind that the output could be heavy for printing if there are lot of parts.

Examples:

```
\dontrun{
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Get upload job status information
 upload_object$info()
}
```

Method `start()`**:** Start the file upload

Usage:

```
Upload$start()
```

Returns: `File` object.

Examples:

```
\dontrun{
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Initialize multipart file upload object
 upload_object$init()

 # Start upload process
 upload_object$start()
}
```

Method `abort()`**:** Abort the multipart upload This call aborts an ongoing upload.

Usage:

```
Upload$abort()
```

Examples:

```
\dontrun{
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )
```

```
 # Abort upload process
 upload_object$abort()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Upload$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Upload$print`
## ------------------------------------------------

## Not run:
 upload_object <- Upload$new(
                   path = "path/to/my/file.txt",
                   project = project_object,
                   auth = auth
                  )

 # Print the upload object information
 upload_object$print(name = name)

## End(Not run)


## ------------------------------------------------
## Method `Upload$init`
## ------------------------------------------------

## Not run:
 upload_object <- Upload$new(
                   path = "path/to/my/file.txt",
                   project = project_object,
                   auth = auth
                  )

 # Initialize multipart file upload object
 upload_object$init()

## End(Not run)

## ------------------------------------------------
## Method `Upload$info`
## ------------------------------------------------

## Not run:
 upload_object <- Upload$new(
```

```
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Get upload job status information
 upload_object$info()

## End(Not run)

## ------------------------------------------------
## Method `Upload$start`
## ------------------------------------------------

## Not run:
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Initialize multipart file upload object
 upload_object$init()

 # Start upload process
 upload_object$start()

## End(Not run)


## ------------------------------------------------
## Method `Upload$abort`
## ------------------------------------------------

## Not run:
 upload_object <- Upload$new(
                    path = "path/to/my/file.txt",
                    project = project_object,
                    auth = auth
                   )

 # Abort upload process
 upload_object$abort()

## End(Not run)
```

User *R6 Class Representing a platform User*

Description

User object containing user information.

Details

This is the main object for Users.

Super class

`sevenbridges2::Item` -> `User`

Public fields

`URL` List of URL endpoints for this resource.

`username` User name.

`email` User's email address.

`first_name` User's first name.

`last_name` User's last name.

`affiliation` The company or the institute the user is affiliated with.

`phone` User's phone number.

`address` User's residential address.

`city` User's city of residence.

`state` User's state of residence.

`country` User's country of residence.

`zip_code` Zip code for the user's residence.

`role` User's role.

`tags` Platform tags associated with the user.

Methods

Public methods:

- `User$new()`
- `User$print()`
- `User$reload()`
- `User$clone()`

Method `new()`**:** Create a new User object.

Usage:

```
User$new(res = NA, ...)
```

Arguments:

`res` Response containing User object information.

`...` Other response arguments.

Method `print()`**:** Print user information as bullet list.

Usage:

```
User$print()
```

Method `reload()`**:** Reload User object information.

Usage:

```
User$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `User` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
User$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`Volume` *R6 Class representing a Volume*

Description

R6 Class representing a resource for managing volumes.

Super class

`sevenbridges2::Item` -> `Volume`

Public fields

`URL` List of URL endpoints for this resource.

`id` Volume ID, constructed from `{division}/{volume_name}` or `{volume_owner}/{volume_name}`.

`name` The name of the volume. It must be unique from all other volumes for this user. Required if `from_path` parameter is not provided.

`description` The description of the volume.

`access_mode` Signifies whether this volume should be used for read-write (RW) or read-only (RO) operations. The access mode is consulted independently of the credentials granted to Seven Bridges when the volume was created, so it is possible to use a read-write credentials to register both read-write and read-only volumes using it. Default: `"RW"`.

`service` This object in form of string more closely describes the mapping of the volume to the cloud service where the data is stored.

`created_on` The date and time this volume was created.

`modified_on` The date and time this volume was last modified.

`active` If a volume is deactivated, this field will be set to `FALSE`.

Methods

Public methods:

- Volume$new()
- Volume$print()
- Volume$reload()
- Volume$update()
- Volume$deactivate()
- Volume$reactivate()
- Volume$delete()
- Volume$list_contents()
- Volume$get_file()
- Volume$list_members()
- Volume$add_member()
- Volume$add_member_team()
- Volume$add_member_division()
- Volume$remove_member()
- Volume$get_member()
- Volume$modify_member_permissions()
- Volume$list_imports()
- Volume$list_exports()
- Volume$clone()

Method `new()`**:** Create a new Volume object.

Usage:

```
Volume$new(res = NA, ...)
```

Arguments:

`res` Response containing Volume object info.

`...` Other response arguments.

Method `print()`**:** Print method for Volume class.

Usage:

```
Volume$print()
```

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print volume object
```

```
 volume_object$print()
}
```

Method `reload()`**:** Reload Volume object information.

Usage:

```
Volume$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Volume` object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload volume object
 volume_object$reload()
}
```

Method `update()`**:** Update a volume. This function updates the details of a specific volume.

Usage:

```
Volume$update(description = NULL, access_mode = NULL, service = NULL)
```

Arguments:

`description` The new description of the volume.

`access_mode` Signifies whether this volume should be used for read-write (RW) or read-only (RO) operations. The access mode is consulted independently of the credentials granted to Seven Bridges when the volume was created, so it is possible to use a read-write credentials to register both read-write and read-only volumes using it. Default: `"RW"`.

`service` This object in form of string more closely describes the mapping of the volume to the cloud service where the data is stored.

Returns: `Volume` object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
```

```
                    response = attr(x, "response")
                   )

 # Update volume object
 volume_object$update(description = description)
}
```

Method `deactivate()`**:** Deactivate volume. Once deactivated, you cannot import from, export to, or browse within a volume. As such, the content of the files imported from this volume will no longer be accessible on the Platform. However, you can update the volume and manage members. Note that you cannot deactivate the volume if you have running imports or exports unless you force the operation using the query parameter force=TRUE. Note that to delete a volume, first you must deactivate it and delete all files which have been imported from the volume to the Platform.

Usage:

```
Volume$deactivate(...)
```

Arguments:

`...` Other query parameters or arguments that can be passed to core `api()` function like 'force'. Use it within query parameter, like `query = list(force = TRUE)`.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Deactivate volume
 volume_object$deactivate()
}
```

Method `reactivate()`**:** Reactivate volume. This function reactivates the previously deactivated volume by updating the `active` field of the volume to `TRUE`.

Usage:

```
Volume$reactivate(...)
```

Arguments:

`...` Other query parameters or arguments that can be passed to core `api()` function like 'force'. Use it within query parameter, like `query = list(force = TRUE)`.

Returns: `Volume` object.

Examples:

```
\dontrun{
# x is API response when volume is requested
```

```
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Deactivate volume
 volume_object$deactivate()

 # Reactivate volume
 volume_object$reactivate()
}
```

Method `delete()`**:** Delete volume. This call deletes a volume you've created to refer to storage on Amazon Web Services, Google Cloud Storage, Azure or Ali cloud. To be able to delete a volume, you first need to deactivate it and then delete all files on the Platform that were previously imported from the volume.

Usage:

```
Volume$delete()
```

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete volume
 volume_object$delete()
}
```

Method `list_contents()`**:** List volume contents. This call lists the contents of a specific volume.

Usage:

```
Volume$list_contents(
  prefix = NULL,
  limit = getOption("sevenbridges2")$limit,
  link = NULL,
  continuation_token = NULL,
  ...
)
```

Arguments:

`prefix` This is parent parameter in volume context. If specified, the content of the parent directory on the current volume is listed.

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`link` Link to use in the next chunk of results. Contains limit and continuation_token. If provided it will overwrite other arguments' values passed.

`continuation_token` Continuation token received to use for next chunk of results. Behaves similarly like offset parameter.

`...` Other arguments that can be passed to core `api()` function like 'fields' for example. With fields parameter you can specify a subset of fields to include in the response. You can use: `href`, `location`, `volume`, `type`, `metadata`, `_all`. Default: `_all`.

Returns: `VolumeContentCollection` object containing list of `VolumeFile` and `VolumePrefix` objects.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume contents
 volume_object$list_contents()
}
```

Method `get_file()`**:** Get volume file information. This function returns the specific Volume File.

Usage:

```
Volume$get_file(location = NULL, link = NULL, ...)
```

Arguments:

`location` Volume file id, which is represented as file location.

`link` Link to the file resource received from listing volume's contents. Cannot be used together with location.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `VolumeFile` object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
```

```
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get volume file
 volume_object$get_file(location = location)
}
```

Method `list_members()`**:** List members of a volume. This function returns the members of a specific volume.

Usage:

```
Volume$list_members(
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other parameters that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` containing `Member` objects.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume members
 volume_object$list_members()
}
```

Method `add_member()`**:** Add member to a volume. This function adds members to the specified volume.

Usage:

```
Volume$add_member(
  user,
  permissions = list(read = TRUE, copy = FALSE, write = FALSE, admin = FALSE)
)
```

Arguments:

`user` The Seven Bridges Platform username of the person you want to add to the volume or object of class Member containing user's username.

`permissions` List of permissions granted to the user being added. Permissions include listing the contents of a volume, importing files from the volume to the Platform, exporting files from the Platform to the volume, and admin privileges.
It can contain fields: 'read', 'copy', 'write' and 'admin' with logical fields - TRUE if certain permission is allowed to the user, or FALSE if it's not. Example:

```
permissions = list(read = TRUE, copy = TRUE, write = FALSE,
admin = FALSE)
```

Returns: Member object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add volume member
 volume_object$add_member(
                user = user,
                permissions = list(read = TRUE, copy = FALSE)
              )
}
```

Method `add_member_team()`**:** Add a specific team as a member to a volume. Only Enterprise users that are part of some division can add teams to a volume created within that division.

Usage:

```
Volume$add_member_team(
  team,
  permissions = list(read = TRUE, copy = FALSE, write = FALSE, admin = FALSE)
)
```

Arguments:

`team` The Seven Bridges Platform ID of a team you want to add to the volume or object of class Team containing team's ID. Team must be created within a division where the volume is created too.

`permissions` List of permissions granted to the team being added. Permissions include listing the contents of a volume, importing files from the volume to the Platform, exporting files from the Platform to the volume, and admin privileges.
It can contain fields: 'read', 'copy', 'write' and 'admin' with logical fields - TRUE if certain permission is allowed to the team, or FALSE if it's not. Example:

```
permissions = list(read = TRUE, copy = TRUE, write = FALSE,
admin = FALSE)
```

Returns: Member object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add volume member
 volume_object$add_member_team(
                team = <team-id>,
                permissions = list(read = TRUE, copy = FALSE)
              )
}
```

Method `add_member_division()`**:** Add a specific division as a member to a volume. Only Enterprise users (with Enterprise accounts) can add divisions to a volume that is created with that Enterprise account (not within other divisions).

Usage:

```
Volume$add_member_division(
  division,
  permissions = list(read = TRUE, copy = FALSE, write = FALSE, admin = FALSE)
)
```

Arguments:

`division` The Seven Bridges Platform ID of a division you want to add to the volume or object of class Division containing division's ID.

`permissions` List of permissions granted to the division being added. Permissions include listing the contents of a volume, importing files from the volume to the Platform, exporting files from the Platform to the volume, and admin privileges.
It can contain fields: 'read', 'copy', 'write' and 'admin' with logical fields - TRUE if certain permission is allowed to the division, or FALSE if it's not. Example:

```
permissions = list(read = TRUE, copy = TRUE, write = FALSE,
admin = FALSE)
```

Returns: Member object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
```

```
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add volume member
 volume_object$add_member_division(
                division = <division-id>,
                permissions = list(read = TRUE, copy = FALSE)
               )
}
```

Method `remove_member()`**:** Remove member from a volume. This function removes members from the specified volume.

Usage:

```
Volume$remove_member(member)
```

Arguments:

`member` The Seven Bridges Platform username of the person you want to remove from the volume, or team ID or division ID (for Enterprise users only) or object of class Member containing member's ID.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Remove volume member
 volume_object$remove_member(member = member)
}
```

Method `get_member()`**:** Get member's details. This function returns member's information.

Usage:

```
Volume$get_member(member, ...)
```

Arguments:

`member` The Seven Bridges Platform username of the person you want to get information about, or team ID or division ID (for Enterprise users only) or object of class Member containing member's ID.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Member` object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get volume member
 volume_object$get_member(member = member)
}
```

Method `modify_member_permissions()`**:** Modify volume member's permissions. This function modifies the permissions for a member of a specific volume. Note that this does not overwrite all previously set permissions for the member.

Usage:

```
Volume$modify_member_permissions(
  member,
  permissions = list(read = TRUE, copy = FALSE, write = FALSE, admin = FALSE)
)
```

Arguments:

`member` The Seven Bridges Platform username of the person you want to modify permissions for or team ID or division ID (for Enterprise users only) or object of class Member containing member's ID.

`permissions` List of specific (or all) permissions you want to update for the member of the volume. Permissions include listing the contents of a volume, importing files from the volume to the Platform, exporting files from the Platform to the volume, and admin privileges. It can contain fields: 'read', 'copy', 'write' and 'admin' with logical fields - TRUE if certain permission is allowed to the user, or FALSE if it's not. Example:

```
permissions = list(read = TRUE, copy = TRUE, write = FALSE,
admin = FALSE)
```

Returns: `Permission` object.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Modify volume member permissions
 volume_object$modify_member_permissions(
```

```
                    member = member,
                    permission = list(read = TRUE, copy = TRUE)
                  )
}
```

Method `list_imports()`**:** This call lists import jobs initiated by particular user from this volume.

Usage:

```
Volume$list_imports(
  project = NULL,
  state = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`project` String project id or Project object. List all volume imports to this project. Optional.

`state` String. The state of the import job. Possible values are:

- `PENDING`: the import is queued;
- `RUNNING`: the import is running;
- `COMPLETED`: the import has completed successfully;
- `FAILED`: the import has failed.

Example: `state = c("RUNNING", "FAILED")`

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` containing list of `Import` job objects.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume imports
 volume_object$list_imports(
                    project = project,
                    state = c("RUNNING", "FAILED")
```

```
                    )
}
```

Method `list_exports()`**:** This call lists export jobs initiated by a user into this volume. Note that when you export a file from a project on the Platform into a volume, you write to your cloud storage bucket.

Usage:

```
Volume$list_exports(
  state = NULL,
  limit = getOption("sevenbridges2")$limit,
  offset = getOption("sevenbridges2")$offset,
  ...
)
```

Arguments:

`state` The state of the export job. Possible values are:

- `PENDING`: the export is queued;
- `RUNNING`: the export is running;
- `COMPLETED`: the export has completed successfully;
- `FAILED`: the export has failed.

Example: `state = c("RUNNING", "FAILED")`

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`offset` The zero-based starting index in the entire collection of the first item to return. The default value is `0`. This is a pagination-specific attribute.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Collection` containing list of `Export` job objects.

Examples:

```
\dontrun{
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume exports
 volume_object$list_exports(state = c("RUNNING", "FAILED"))
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Volume$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Volume$print`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print volume object
 volume_object$print()

## End(Not run)


## ------------------------------------------------
## Method `Volume$reload`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload volume object
 volume_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `Volume$update`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
```

```
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Update volume object
 volume_object$update(description = description)

## End(Not run)


## ------------------------------------------------
## Method `Volume$deactivate`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Deactivate volume
 volume_object$deactivate()

## End(Not run)


## ------------------------------------------------
## Method `Volume$reactivate`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Deactivate volume
 volume_object$deactivate()

 # Reactivate volume
 volume_object$reactivate()

## End(Not run)


## ------------------------------------------------
```

```
## Method `Volume$delete`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Delete volume
 volume_object$delete()

## End(Not run)


## ------------------------------------------------
## Method `Volume$list_contents`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume contents
 volume_object$list_contents()

## End(Not run)


## ------------------------------------------------
## Method `Volume$get_file`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get volume file
 volume_object$get_file(location = location)
```

```
## End(Not run)


## ------------------------------------------------
## Method `Volume$list_members`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume members
 volume_object$list_members()

## End(Not run)


## ------------------------------------------------
## Method `Volume$add_member`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add volume member
 volume_object$add_member(
                user = user,
                permissions = list(read = TRUE, copy = FALSE)
              )

## End(Not run)


## ------------------------------------------------
## Method `Volume$add_member_team`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
```

```
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add volume member
 volume_object$add_member_team(
                team = <team-id>,
                permissions = list(read = TRUE, copy = FALSE)
               )

## End(Not run)


## ------------------------------------------------
## Method `Volume$add_member_division`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Add volume member
 volume_object$add_member_division(
                division = <division-id>,
                permissions = list(read = TRUE, copy = FALSE)
               )

## End(Not run)


## ------------------------------------------------
## Method `Volume$remove_member`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Remove volume member
 volume_object$remove_member(member = member)

## End(Not run)
```

```
## ------------------------------------------------
## Method `Volume$get_member`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get volume member
 volume_object$get_member(member = member)

## End(Not run)


## ------------------------------------------------
## Method `Volume$modify_member_permissions`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Modify volume member permissions
 volume_object$modify_member_permissions(
                    member = member,
                    permission = list(read = TRUE, copy = TRUE)
                  )

## End(Not run)


## ------------------------------------------------
## Method `Volume$list_imports`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
```

```
                    )

 # List volume imports
 volume_object$list_imports(
                     project = project,
                     state = c("RUNNING", "FAILED")
                    )

## End(Not run)


## ------------------------------------------------
## Method `Volume$list_exports`
## ------------------------------------------------

## Not run:
# x is API response when volume is requested
volume_object <- Volume$new(
                     res = x,
                     href = x$href,
                     auth = auth,
                     response = attr(x, "response")
                    )

 # List volume exports
 volume_object$list_exports(state = c("RUNNING", "FAILED"))

## End(Not run)
```

`VolumeContentCollection`
R6 Class representing a VolumeContentCollection

Description

R6 Class representing a resource for managing volume content collections.

Super class

`sevenbridges2::Collection` -> `VolumeContentCollection`

Public fields

`prefixes` Prefixes on the volume, returned in API response.

Methods

Public methods:

- VolumeContentCollection$new()
- VolumeContentCollection$print()
- VolumeContentCollection$next_page()
- VolumeContentCollection$prev_page()
- VolumeContentCollection$all()
- VolumeContentCollection$clone()

Method `new()`**:** Create new VolumeContentCollection object.

Usage:

```
VolumeContentCollection$new(res = NA, ...)
```

Arguments:

`res` Response containing VolumeContentCollection object fields.

`...` Other response arguments.

Method `print()`**:** Print method for VolumeContentCollection class.

Usage:

```
VolumeContentCollection$print(n = 10)
```

Arguments:

`n` Number of items to print in console.

Examples:

```
\dontrun{
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                    res = x,
                    href = x$href,
                    links = x$links,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print volume content collection object
 vol_con_col_object$print()
}
```

Method `next_page()`**:** Return next page of results.

Usage:

```
VolumeContentCollection$next_page(...)
```

Arguments:

`...` Other arguments or query parameters that can be passed to the core `api()` function like 'advance_access', 'fields' etc.

Examples:

```
\dontrun{
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                    res = x,
                    href = x$href,
                    links = x$links,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get next page of results
 vol_con_col_object$next_page()
}
```

Method `prev_page()`**:** Return the previous page of results.

Usage:

```
VolumeContentCollection$prev_page()
```

Examples:

```
\dontrun{
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                    res = x,
                    href = x$href,
                    links = x$links,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Get previous page of results
 vol_con_col_object$prev_page()
}
```

Method `all()`**:** Fetches all available items.

Usage:

```
VolumeContentCollection$all(...)
```

Arguments:

`...` Other arguments or query parameters that can be passed to the core `api()` function like 'advance_access', 'fields' etc.

Examples:

```
\dontrun{
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                    res = x,
```

```
                      href = x$href,
                      links = x$links,
                      auth = auth,
                      response = attr(x, "response")
                     )

 # Get all results
 vol_con_col_object$all()
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VolumeContentCollection$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `VolumeContentCollection$print`
## ------------------------------------------------

## Not run:
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                    res = x,
                    href = x$href,
                    links = x$links,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print volume content collection object
 vol_con_col_object$print()

## End(Not run)


## ------------------------------------------------
## Method `VolumeContentCollection$next_page`
## ------------------------------------------------

## Not run:
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                    res = x,
                    href = x$href,
                    links = x$links,
                    auth = auth,
                    response = attr(x, "response")
```

```
                    )

 # Get next page of results
 vol_con_col_object$next_page()

## End(Not run)


## ------------------------------------------------
## Method `VolumeContentCollection$prev_page`
## ------------------------------------------------

## Not run:
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                     res = x,
                     href = x$href,
                     links = x$links,
                     auth = auth,
                     response = attr(x, "response")
                    )

 # Get previous page of results
 vol_con_col_object$prev_page()

## End(Not run)


## ------------------------------------------------
## Method `VolumeContentCollection$all`
## ------------------------------------------------

## Not run:
# x is API response when volume content collection is requested
vol_con_col_object <- VolumeContentCollection$new(
                     res = x,
                     href = x$href,
                     links = x$links,
                     auth = auth,
                     response = attr(x, "response")
                    )

 # Get all results
 vol_con_col_object$all()

## End(Not run)
```

VolumeFile *R6 Class representing a VolumeFile*

Description

R6 Class representing a resource for managing VolumeFile objects.

Super class

`sevenbridges2::Item` -> `VolumeFile`

Public fields

`URL` List of URL endpoints for this resource.

`location` File location on the volume.

`type` Type of storage (cloud provider). Can be one of: `s3`, `gcs`, `azure`, `OSS`.

`volume` Volume id.

`metadata` File metadata, if it exists.

Methods

Public methods:

- `VolumeFile$new()`
- `VolumeFile$print()`
- `VolumeFile$reload()`
- `VolumeFile$import()`
- `VolumeFile$clone()`

Method `new()`**:** Create a new VolumeFile object.

Usage:

```
VolumeFile$new(res = NA, ...)
```

Arguments:

`res` Response containing VolumeFile object info.

`...` Other response arguments.

Method `print()`**:** Print method for VolumeFile class.

Usage:

```
VolumeFile$print()
```

Examples:

```
\dontrun{
# x is API response when volume file is requested
volume_file_object <- VolumeFile$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print volume file object
```

```
 volume_file_object$print()
}
```

Method `reload()`**:** Reload VolumeFile object information.

Usage:

```
VolumeFile$reload(...)
```

Arguments:

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: A `VolumeFile` object.

Examples:

```
\dontrun{
# x is API response when volume file is requested
volume_file_object <- VolumeFile$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload volume file object
 volume_file_object$reload()
}
```

Method `import()`**:** This call lets you queue a job to import this file or folder from a volume into a project on the Platform.
Essentially, you are importing an item from your cloud storage provider (Amazon Web Services, Google Cloud Storage, Azure or Ali Cloud) via the volume onto the Platform.
If successful, an alias will be created on the Platform. Aliases appear on the Platform and can be copied, executed, and modified as such. They refer back to the respective item on the given volume.

Usage:

```
VolumeFile$import(
  destination_project = NULL,
  destination_parent = NULL,
  name = NULL,
  overwrite = FALSE,
  autorename = FALSE,
  ...
)
```

Arguments:

`destination_project` String destination project id or Project object. Not required, but either `destination_project` or `destination_parent` directory must be provided.

`destination_parent` String folder id or File object (with `type = 'FOLDER'`). Not required, but either `destination_project` or `destination_parent` directory must be provided.

`name` The name of the alias to create. This name should be unique to the project.
If the name is already in use in the project, you should use the `overwrite` query parameter in this call to force any item with that name to be deleted before the alias is created. If name is omitted, the alias name will default to the last segment of the complete location (including the prefix) on the volume.

Segments are considered to be separated with forward slashes /. Allowed characters in file names are all alphanumeric and special characters except forward slash /, while folder names can contain alphanumeric and special characters _, - and ..

`overwrite` Set to `TRUE` if you want to overwrite the item if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder's content (files with the same name) will be overwritten, not the whole folder.

`autorename` Set to `TRUE` if you want to automatically rename the item (by prefixing its name with an underscore and number) if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder content will be renamed, not the whole folder.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Import` object.

Examples:

```
\dontrun{
# x is API response when volume file is requested
volume_file_object <- VolumeFile$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Import volume file object
 volume_file_object$import(destination_project = destination_project)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VolumeFile$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `VolumeFile$print`
## ------------------------------------------------

## Not run:
# x is API response when volume file is requested
```

```
volume_file_object <- VolumeFile$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print volume file object
 volume_file_object$print()

## End(Not run)


## ------------------------------------------------
## Method `VolumeFile$reload`
## ------------------------------------------------

## Not run:
# x is API response when volume file is requested
volume_file_object <- VolumeFile$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload volume file object
 volume_file_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `VolumeFile$import`
## ------------------------------------------------

## Not run:
# x is API response when volume file is requested
volume_file_object <- VolumeFile$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Import volume file object
 volume_file_object$import(destination_project = destination_project)

## End(Not run)
```

VolumePrefix *R6 Class representing a VolumePrefix*

Description

R6 Class representing a resource for managing VolumePrefix objects.

Super class

sevenbridges2::Item -> VolumePrefix

Public fields

URL List of URL endpoints for this resource.

prefix File/prefix name on the volume.

volume Volume id.

Methods

Public methods:

- VolumePrefix$new()
- VolumePrefix$print()
- VolumePrefix$reload()
- VolumePrefix$list_contents()
- VolumePrefix$import()
- VolumePrefix$clone()

Method new(): Create a new VolumePrefix object.

Usage:

```
VolumePrefix$new(res = NA, ...)
```

Arguments:

res Response containing VolumePrefix object information.

... Other response arguments.

Method print(): Print method for VolumePrefix class.

Usage:

```
VolumePrefix$print()
```

Examples:

```
\dontrun{
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
```

```
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Print the Volume Prefix object
 volume_prefix_object$print()
}
```

Method `reload()`**:** Reload the VolumePrefix object information.

Usage:

```
VolumePrefix$reload()
```

Examples:

```
\dontrun{
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload volume prefix object
 volume_prefix_object$reload()
}
```

Method `list_contents()`**:** List the contents of a volume folder. This call lists the contents of a specific volume folder.

Usage:

```
VolumePrefix$list_contents(
  limit = getOption("sevenbridges2")$limit,
  continuation_token = NULL,
  ...
)
```

Arguments:

`limit` The maximum number of collection items to return for a single request. Minimum value is `1`. The maximum value is `100` and the default value is `50`. This is a pagination-specific attribute.

`continuation_token` Continuation token received to use for next chunk of results. Behaves similarly like offset parameter.

`...` Other arguments that can be passed to core `api()` function, like 'fields' for example. With fields parameter you can specify a subset of fields to include in the response. You can use: `href`, `location`, `volume`, `type`, `metadata`, `_all`. Default: `_all`.

Returns: `VolumeContentCollection` object containing list of `VolumeFile` and `VolumePrefix` objects.

Examples:

```
\dontrun{
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume prefix object contents
 volume_prefix_object$list_contents()
}
```

Method `import()`**:** This call lets you queue a job to import this file or folder from a volume into a project on the Platform.
Essentially, you are importing an item from your cloud storage provider (Amazon Web Services, Google Cloud Storage, Azure or Ali Cloud) via the volume onto the Platform.
If successful, an alias will be created on the Platform. Aliases appear on the Platform and can be copied, executed, and modified as such. They refer back to the respective item on the given volume.

Usage:

```
VolumePrefix$import(
  destination_project = NULL,
  destination_parent = NULL,
  name = NULL,
  overwrite = FALSE,
  autorename = FALSE,
  preserve_folder_structure = NULL,
  ...
)
```

Arguments:

`destination_project` String destination project id or Project object. Not required, but either `destination_project` or `destination_parent` directory must be provided.

`destination_parent` String folder id or File object (with `type = 'FOLDER'`). Not required, but either `destination_project` or `destination_parent` directory must be provided.

`name` The name of the alias to create. This name should be unique to the project. If the name is already in use in the project, you should use the `overwrite` query parameter in this call to force any item with that name to be deleted before the alias is created. If name is omitted, the alias name will default to the last segment of the complete location (including the prefix) on the volume.

Segments are considered to be separated with forward slashes /. Allowed characters in file names are all alphanumeric and special characters except forward slash (/), while folder names can contain alphanumeric and special characters underscores (_), hyphens (-), and dots (.).

`overwrite` Set to `TRUE` if you want to overwrite the item if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder's content (files with the same name) will be overwritten, not the whole folder.

`autorename` Set to `TRUE` if you want to automatically rename the item (by prefixing its name with an underscore and number) if another one with the same name already exists at the destination. Bear in mind that if used with folders import, the folder content will be renamed, not the whole folder.

`preserve_folder_structure` Set to `TRUE` if you want to keep the exact source folder structure. The default value is `TRUE` if the item being imported is a folder. Should not be used if you are importing a file. Bear in mind that if you use `preserve_folder_structure = FALSE`, that the response will be the parent folder object containing imported files alongside with other files if they exist.

`...` Other arguments that can be passed to core `api()` function like 'fields', etc.

Returns: `Import` object.

Examples:

```
\dontrun{
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume prefix object contents
 volume_prefix_object$import(destination_project = destination_project)
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VolumePrefix$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `VolumePrefix$print`
## ------------------------------------------------

## Not run:
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
```

```
                    response = attr(x, "response")
                   )

 # Print the Volume Prefix object
 volume_prefix_object$print()

## End(Not run)


## ------------------------------------------------
## Method `VolumePrefix$reload`
## ------------------------------------------------

## Not run:
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # Reload volume prefix object
 volume_prefix_object$reload()

## End(Not run)


## ------------------------------------------------
## Method `VolumePrefix$list_contents`
## ------------------------------------------------

## Not run:
# x is API response when volume prefix is requested
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume prefix object contents
 volume_prefix_object$list_contents()

## End(Not run)


## ------------------------------------------------
## Method `VolumePrefix$import`
## ------------------------------------------------

## Not run:
# x is API response when volume prefix is requested
```

```
volume_prefix_object <- VolumePrefix$new(
                    res = x,
                    href = x$href,
                    auth = auth,
                    response = attr(x, "response")
                   )

 # List volume prefix object contents
 volume_prefix_object$import(destination_project = destination_project)

## End(Not run)
```

Volumes *R6 Class representing volumes endpoints*

Description

R6 Class representing volumes resource endpoints.

Super class

sevenbridges2::Resource -> Volumes

Public fields

URL List of URL endpoints for this resource.

Methods

Public methods:

- Volumes$new()
- Volumes$query()
- Volumes$get()
- Volumes$delete()
- Volumes$create_s3_using_iam_user()
- Volumes$create_s3_using_iam_role()
- Volumes$create_google_using_iam_user()
- Volumes$create_google_using_iam_role()
- Volumes$create_azure()
- Volumes$create_ali_oss()
- Volumes$clone()

Method new(): Create a new Volumes object.

Usage:

```
Volumes$new(...)
```

Arguments:

... Other response arguments.

Method `query()`**:** This call lists all the volumes you've registered.

Usage:

```
Volumes$query(...)
```

Arguments:

... Other arguments that can be passed to core `api()` function like 'limit', 'offset', 'fields', etc.

Returns: `Collection` of `Volume` objects.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 # Query volumes
 volumes_object$query()
}
```

Method `get()`**:** This call returns details of the specified volume. The volume is referred to by its ID, which you can obtain by making the call to list all the volumes you've registered.

Usage:

```
Volumes$get(id)
```

Arguments:

`id` The Volume ID consists of volume owner's name (for enterprise users) and volume name in form `{volume_owner}/{volume_name}`, or division name (if user belongs to some division) and volume name in form `{division}/{volume_name}`. You can also get the Volume ID for a volume by making the call to list all volumes you've registered.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 # Get volumes
 volumes_object$get(id = id)
}
```

Method `delete()`**:** This call deletes a volume you've created to refer to storage on Amazon Web Services or Google Cloud Storage. To be able to delete a volume, you first need to deactivate it and then delete all files on the Platform that were previously imported from the volume.

Volumes are specified by their IDs, which you can obtain by using `Volumes$query()` to list files or by getting a single file using `Volumes$get()`.

Usage:

```
Volumes$delete(volume, ...)
```

Arguments:

`volume` `Volume` object or volume ID.

`...` Other arguments that can be passed to core `api()` function as 'fields', etc.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 # Get volumes
 volumes_object$delete(volume = volume)
}
```

Method `create_s3_using_iam_user()`**:** Create new volume to connect to your s3 bucket on AWS cloud. Volumes authorize the Platform to access and query objects on a specified cloud storage (Amazon Web Services, Google Cloud Storage, Azure or Ali cloud) on your behalf. This function uses IAM User credentials to connect to your s3 bucket.

Read more about volume creation in our API documentation.

Usage:

```
Volumes$create_s3_using_iam_user(
  name = NULL,
  access_mode = "RW",
  description = NULL,
  prefix = NULL,
  bucket = NULL,
  endpoint = "s3.amazonaws.com",
  access_key_id = NULL,
  secret_access_key = NULL,
  properties = list(sse_algorithm = "AES256"),
  from_path = NULL
)
```

Arguments:

`name` The name of the volume. It must be unique from all other volumes for this user. Required if from_path parameter is not provided.

`access_mode` Signifies whether this volume should be used for read-write (RW) or read-only (RO) operations. The access mode is consulted independently of the credentials granted to Seven Bridges when the volume was created, so it is possible to use a read-write credentials to register both read-write and read-only volumes using it. Default: `"RW"`.

`description` An optional description of this volume.

`prefix` A service-specific string prefix to append to all objects created in this volume. If the service supports folders, and this prefix includes them, the API will attempt to create any missing folders when it outputs a file.

`bucket` The name of the AWS S3 bucket you wish to register as a volume. Required if `from_path` parameter is not provided.

`endpoint` AWS API endpoint to use when accessing this bucket. Default: `s3.amazonaws.com`.

`access_key_id` AWS access key ID in form of string of the IAM user shared with Seven Bridges to access this bucket. Required if `from_path` parameter is not provided.

`secret_access_key` AWS secret access key in form of string of the IAM user shared with Seven Bridges to access this bucket. Required if `from_path` parameter is not provided.

`properties` Named list containing the properties of a specific service. These values set the defaults for operations performed with this volume. Individual operations can override these defaults by providing a custom properties object. For AWS S3, there are:

- `sse_algorithm` - S3 server-side encryption to use when exporting to this bucket. Supported values: `AES256` (SSE-S3 encryption), `aws:kms`, `null` (no server-side encryption). Default: `AES256`.
- `sse_aws_kms_key_id`: Applies to type: `s3`. If AWS KMS encryption is used, this should be set to the required KMS key. If not set and `aws:kms` is set as `sse_algorithm`, default KMS key is used.
- `aws_canned_acl`: S3 canned ACL to apply on the object on during export. Supported values: any one of S3 canned ACLs; `null` (do not apply canned ACLs). Default: `null`.

`from_path` Path to JSON configuration file containing all required information for registering a volume. If provided, it will overwrite all previous parameters set.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 # Create new AWS Volume (IAM User)
 aws_iam_user_volume <- volumes_object$create_s3_using_iam_user(
    name = "my_new_aws_user_volume",
    bucket = "<bucket-name>",
    description = "AWS IAM User Vol",
    access_key_id = "<access-key>",
    secret_access_key = "<secret-access-key>"
 )
}
```

Method `create_s3_using_iam_role()`**:** Create new volume to connect to your s3 bucket on AWS cloud. Volumes authorize the Platform to access and query objects on a specified cloud storage (Amazon Web Services, Google Cloud Storage, Azure or Ali cloud) on your behalf. This function uses IAM Role credentials to connect to your s3 bucket. In order to use these credentials, user must have specific user tag enabled by Support team.

Read more about volume creation in our API documentation.

Usage:

```
Volumes$create_s3_using_iam_role(
  name = NULL,
  access_mode = "RW",
```

```
  description = NULL,
  prefix = NULL,
  bucket = NULL,
  endpoint = "s3.amazonaws.com",
  role_arn = NULL,
  external_id = NULL,
  properties = list(sse_algorithm = "AES256"),
  from_path = NULL
)
```

Arguments:

`name` The name of the volume. It must be unique from all other volumes for this user. Required if from_path parameter is not provided.

`access_mode` Signifies whether this volume should be used for read-write (RW) or read-only (RO) operations. The access mode is consulted independently of the credentials granted to Seven Bridges when the volume was created, so it is possible to use a read-write credentials to register both read-write and read-only volumes using it. Default: `"RW"`.

`description` An optional description of this volume.

`prefix` A service-specific string prefix to append to all objects created in this volume. If the service supports folders, and this prefix includes them, the API will attempt to create any missing folders when it outputs a file.

`bucket` The name of the AWS S3 bucket you wish to register as a volume. Required if `from_path` parameter is not provided.

`endpoint` AWS API endpoint to use when accessing this bucket. Default: `s3.amazonaws.com`.

`role_arn` The ARN (Amazon Resource Name) of your role that is used to connect your S3 bucket. Required if `from_path` parameter is not provided.

`external_id` Optional information that you can use in an IAM role trust policy to designate who can assume the role. Must be provided if it is configured in your role trust policy on AWS. Required if `from_path` parameter is not provided.

`properties` Named list containing the properties of a specific service. These values set the defaults for operations performed with this volume. Individual operations can override these defaults by providing a custom properties object. For AWS S3, there are:

- `sse_algorithm` - S3 server-side encryption to use when exporting to this bucket. Supported values: `AES256` (SSE-S3 encryption), `aws:kms`, `null` (no server-side encryption). Default: `AES256`.
- `sse_aws_kms_key_id`: Applies to type: `s3`. If AWS KMS encryption is used, this should be set to the required KMS key. If not set and `aws:kms` is set as `sse_algorithm`, default KMS key is used.
- `aws_canned_acl`: S3 canned ACL to apply on the object on during export. Supported values: any one of S3 canned ACLs; `null` (do not apply canned ACLs). Default: `null`.

`from_path` Path to JSON configuration file containing all required information for registering a volume. If provided, it will overwrite all previous parameters set.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)
```

```
 # Create new AWS Volume (IAM Role)
 aws_iam_role_volume <- volumes_object$create_s3_using_iam_role(
    name = "my_new_aws_user_volume",
    bucket = "<bucket-name>",
    description = "AWS IAM Role Vol",
    role_arn = "<role-arn-key>",
    external_id = "<external-id>"
 )
}
```

Method `create_google_using_iam_user()`**:** Create new volume to connect to your bucket on GCS. Volumes authorize the Platform to access and query objects on a specified cloud storage (Amazon Web Services, Google Cloud Storage, Azure or Ali cloud) on your behalf. This function uses IAM User credentials to connect with your GCS bucket.

Read more about volume creations in our API documentation.

Usage:

```
Volumes$create_google_using_iam_user(
  name = NULL,
  access_mode = "RW",
  description = NULL,
  prefix = NULL,
  bucket = NULL,
  root_url = "https://www.googleapis.com",
  client_email = NULL,
  private_key = NULL,
  properties = NULL,
  from_path = NULL
)
```

Arguments:

`name` The name of the volume. It must be unique from all other volumes for this user. Required if `from_path` parameter is not provided.

`access_mode` Signifies whether this volume should be used for read-write (RW) or read-only (RO) operations. The access mode is consulted independently of the credentials granted to Seven Bridges when the volume was created, so it is possible to use a read-write credentials to register both read-write and read-only volumes using it. Default: `"RW"`.

`description` An optional description of this volume.

`prefix` A service-specific string prefix to append to all objects created in this volume. If the service supports folders, and this prefix includes them, the API will attempt to create any missing folders when it outputs a file.

`bucket` The name of the GCS bucket you wish to register as a volume. Required if `from_path` parameter is not provided.

`root_url` Google Cloud Storage API endpoint for accessing this bucket. Default: `https://www.googleapis.com`.

`client_email` The client email address for the Google Cloud service account to use for operations on this bucket. This can be found in the JSON containing your service account credentials. Required if `from_path` parameter is not provided.

`private_key` Google Cloud Platform private key. Required if `from_path` parameter is not provided.

`properties` Named list containing the properties of a specific service. These values set the defaults for operations performed with this volume. Individual operations can override these defaults by providing a custom properties object.

`from_path` Path to JSON configuration file containing all required information for registering a volume. If provided, it will overwrite all previous parameters set.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 #  Create Google cloud volume using IAM User authentication type
 gc_iam_user_volume <- volumes_object$create_google_using_iam_user(
    name = "my_new_gc_user_volume",
    access_mode = "RW",
    bucket = "<bucket-name>",
    description = "GC IAM User volume",
    client_email = "<client_email>",
    private_key = "<private_key-string>"
 )
}
```

Method `create_google_using_iam_role()`**:** Create new volume to connect to your bucket on GCS. Volumes authorize the Platform to access and query objects on a specified cloud storage (Amazon Web Services, Google Cloud Storage, Azure or Ali cloud) on your behalf. This function uses IAM Role credentials to connect to your GCS bucket. In order to use these credentials, user must have specific user tag enabled by Support team.

Read more about volume creations in our API documentation.

Usage:

```
Volumes$create_google_using_iam_role(
  name = NULL,
  access_mode = "RW",
  description = NULL,
  prefix = NULL,
  bucket = NULL,
  root_url = "https://www.googleapis.com",
  configuration = NULL,
  properties = NULL,
  from_path = NULL
)
```

Arguments:

`name` The name of the volume. It must be unique from all other volumes for this user. Required if from_path parameter is not provided.

`access_mode` Signifies whether this volume should be used for read-write (RW) or read-only (RO) operations. The access mode is consulted independently of the credentials granted to Seven Bridges when the volume was created, so it is possible to use a read-write credentials to register both read-write and read-only volumes using it. Default: `"RW"`.

`description` An optional description of this volume.

`prefix` A service-specific string prefix to append to all objects created in this volume. If the service supports folders, and this prefix includes them, the API will attempt to create any missing folders when it outputs a file.

`bucket` The name of the GCS bucket you wish to register as a volume. Required if `from_path` parameter is not provided.

`root_url` Google Cloud Storage API endpoint for accessing this bucket. Default: `https://www.googleapis.com`.

`configuration` Connection configuration parameters in JSON format downloaded from the Google Cloud Console once prerequisites have been set up. Could be provided as a named list, or as path to the downloaded JSON file.

`properties` Named list containing the properties of a specific service. These values set the defaults for operations performed with this volume. Individual operations can override these defaults by providing a custom properties object.

`from_path` Path to JSON configuration file containing all required information for registering a volume. If provided, it will overwrite all previous parameters set.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 #  Create Google cloud volume using IAM User authentication type
 gc_iam_role_volume <- volumes_object$create_google_using_iam_role(
    name = "my_new_gc_role_volume",
    access_mode = "RO",
    bucket = "<bucket-name>",
    description = "GC IAM Role volume",
    configuration = list(
        type = "<type-name>",
        audience = "<audience-link>",
        subject_token_type = "<subject_token_type>",
      service_account_impersonation_url = "<service_account_impersonation_url>",
        token_url = "<token_url>",
        credential_source = list(
           environment_id = "<environment_id>",
           region_url = "<region_url>",
           url = "<url>",
          regional_cred_verification_url = "<regional_cred_verification_url>"
         )
   )
```

```
 )
}
```

Method `create_azure()`**:** This call creates a new volume by attaching a Microsoft Azure storage container to the Platform.

Usage:

```
Volumes$create_azure(
  name = NULL,
  description = NULL,
  endpoint = NULL,
  storage_account = NULL,
  container = NULL,
  prefix = NULL,
  tenant_id = NULL,
  client_id = NULL,
  client_secret = NULL,
  resource_id = NULL,
  from_path = NULL
)
```

Arguments:

`name` The name of the volume. It must be unique from all other volumes for this user. Required if `from_path` parameter is not provided.

`description` An optional description of this volume.

`endpoint` Specify a Microsoft Azure endpoint, only if you are using an endpoint that is different from the default one `https://(serviceaccount).blob.core.windows.net`. To make a non-default endpoint work with the Platform, please first make sure it is supported by Seven Bridges.

`storage_account` The name of the storage account that holds the container you want to attach as a volume.

`container` The name of the container that you want to attach as a Volume.

`prefix` A service-specific string prefix to append to all objects created in this volume. If the service supports folders, and this prefix includes them, the API will attempt to create any missing folders when it outputs a file.

`tenant_id` Directory (tenant) ID of the application you created on the Azure Portal for the purpose of attaching your storage container.

`client_id` Application (client) ID of the application you created on the Azure Portal for the purpose of attaching your storage container.

`client_secret` Value of the client secret you created on the Azure Portal for the purpose of attaching your storage container.

`resource_id` Resource ID of the Azure storage account. To get it, go to the Azure Portal, open the storage account's Overview page and click JSON View.

`from_path` JSON configuration file containing all required information for registering a volume.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 # Create Azure volume
 azure_volume <- volumes_object$create_azure(
   name = "my_new_azure_volume",
   description = "Azure volume",
   endpoint = "<endpoint>",
   container = "<bucket-name",
   storage_account = "<storage_account-name>",
   tenant_id = "<tenant_id>",
   client_id = "<client_id>",
   client_secret = "<client_secret>",
   resource_id = "<resource_id>"
 )
}
```

Method `create_ali_oss()`**:** Create new volume to connect to your bucket on ALI (OSS) platform.

Usage:

```
Volumes$create_ali_oss(
  name = NULL,
  description = NULL,
  endpoint = NULL,
  bucket = NULL,
  prefix = NULL,
  access_key_id = NULL,
  secret_access_key = NULL,
  properties = NULL,
  from_path = NULL
)
```

Arguments:

`name` The name of the volume. It must be unique from all other volumes for this user. Required if from_path parameter is not provided.

`description` An optional description of this volume.

`endpoint` Specify an Ali Cloud endpoint.

`bucket` The name of the ALI(OSS) bucket you wish to register as a volume. Required if `from_path` parameter is not provided.

`prefix` A service-specific string prefix to append to all objects created in this volume. If the service supports folders, and this prefix includes them, the API will attempt to create any missing folders when it outputs a file.

`access_key_id` ALI(OSS) access key ID of the user shared with Seven Bridges to access this bucket. Required if `from_path` parameter is not provided.

`secret_access_key` ALI(OSS) secret access key of the user shared with Seven Bridges to access this bucket. Required if `from_path` parameter is not provided.

`properties` Named list containing the properties of a specific service. These values set the defaults for operations performed with this volume. Individual operations can override these defaults by providing a custom properties object.

`from_path` JSON configuration file containing all required information for registering a volume.

Returns: `Volume` object.

Examples:

```
\dontrun{
 volumes_object <- Volumes$new(auth = auth)

 # Create Ali cloud volume
ali_volume <- volumes_object$create_ali_oss(
  name = "my_new_azure_volume",
  description = "Ali volume",
  endpoint = "<endpoint>",
  bucket = "<bucket-name",
  access_key_id = "<access_key_id>",
  secret_access_key = "<secret_access_key>"
 )
}
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Volumes$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Volumes$query`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 # Query volumes
 volumes_object$query()

## End(Not run)


## ------------------------------------------------
## Method `Volumes$get`
## ------------------------------------------------

## Not run:
```

```
 volumes_object <- Volumes$new(auth = auth)

 # Get volumes
 volumes_object$get(id = id)

## End(Not run)


## ------------------------------------------------
## Method `Volumes$delete`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 # Get volumes
 volumes_object$delete(volume = volume)

## End(Not run)


## ------------------------------------------------
## Method `Volumes$create_s3_using_iam_user`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 # Create new AWS Volume (IAM User)
 aws_iam_user_volume <- volumes_object$create_s3_using_iam_user(
    name = "my_new_aws_user_volume",
    bucket = "<bucket-name>",
    description = "AWS IAM User Vol",
    access_key_id = "<access-key>",
    secret_access_key = "<secret-access-key>"
 )

## End(Not run)


## ------------------------------------------------
## Method `Volumes$create_s3_using_iam_role`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 # Create new AWS Volume (IAM Role)
 aws_iam_role_volume <- volumes_object$create_s3_using_iam_role(
    name = "my_new_aws_user_volume",
    bucket = "<bucket-name>",
    description = "AWS IAM Role Vol",
    role_arn = "<role-arn-key>",
```

```
    external_id = "<external-id>"
 )

## End(Not run)


## ------------------------------------------------
## Method `Volumes$create_google_using_iam_user`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 #  Create Google cloud volume using IAM User authentication type
 gc_iam_user_volume <- volumes_object$create_google_using_iam_user(
    name = "my_new_gc_user_volume",
    access_mode = "RW",
    bucket = "<bucket-name>",
    description = "GC IAM User volume",
    client_email = "<client_email>",
    private_key = "<private_key-string>"
 )

## End(Not run)


## ------------------------------------------------
## Method `Volumes$create_google_using_iam_role`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 #  Create Google cloud volume using IAM User authentication type
 gc_iam_role_volume <- volumes_object$create_google_using_iam_role(
    name = "my_new_gc_role_volume",
    access_mode = "RO",
    bucket = "<bucket-name>",
    description = "GC IAM Role volume",
    configuration = list(
        type = "<type-name>",
        audience = "<audience-link>",
        subject_token_type = "<subject_token_type>",
        service_account_impersonation_url = "<service_account_impersonation_url>",
        token_url = "<token_url>",
        credential_source = list(
           environment_id = "<environment_id>",
           region_url = "<region_url>",
           url = "<url>",
           regional_cred_verification_url = "<regional_cred_verification_url>"
         )
   )
 )
```

```
## End(Not run)


## ------------------------------------------------
## Method `Volumes$create_azure`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 # Create Azure volume
 azure_volume <- volumes_object$create_azure(
   name = "my_new_azure_volume",
   description = "Azure volume",
   endpoint = "<endpoint>",
   container = "<bucket-name",
   storage_account = "<storage_account-name>",
   tenant_id = "<tenant_id>",
   client_id = "<client_id>",
   client_secret = "<client_secret>",
   resource_id = "<resource_id>"
 )

## End(Not run)


## ------------------------------------------------
## Method `Volumes$create_ali_oss`
## ------------------------------------------------

## Not run:
 volumes_object <- Volumes$new(auth = auth)

 # Create Ali cloud volume
ali_volume <- volumes_object$create_ali_oss(
  name = "my_new_azure_volume",
  description = "Ali volume",
  endpoint = "<endpoint>",
  bucket = "<bucket-name",
  access_key_id = "<access_key_id>",
  secret_access_key = "<secret_access_key>"
 )

## End(Not run)
```

Index