Package 'sevenbridges'

November 12, 2019

Type Package

Title Seven Bridges Platform API Client and Common Workflow Language Tool Builder in R

Version 1.16.0

Maintainer Nan Xiao <nan.xiao@sevenbridges.com>

Description R client and utilities for Seven Bridges platform API, from Cancer Genomics Cloud to other Seven Bridges supported platforms.

License Apache License 2.0 | file LICENSE

VignetteBuilder knitr

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

BugReports https://github.com/sbg/sevenbridges-r/issues

biocViews Software, DataImport, ThirdPartyClient

Depends methods, utils, stats

Imports httr, jsonlite, yaml, objectProperties, stringr, S4Vectors, docopt, curl, uuid, dplyr

Suggests knitr, rmarkdown, testthat, readr

NeedsCompilation no

Encoding UTF-8

RoxygenNote 6.1.1

Collate sevenbridges-package.R class-all.R api-http.R api-utils.R api-misc.R class-filesystem.R class-cwl.R class-auth-utils.R class-auth.R class-item.R class-meta.R class-ratelimit.R class-user.R class-billing.R class-member.R class-project.R class-files.R class-upload.R class-app.R class-tool.R class-flow.R class-task.R class-volume.R class-division.R class-team.R class-teammember.R class-marker.R misc-uploader.R misc-handler.R misc-lift.R zzz.R

git_url https://git.bioconductor.org/packages/sevenbridges

git_branch RELEASE_3_10

git_last_commit 0d9ee57

git_last_commit_date 2019-10-29

Date/Publication 2019-11-11

Author Nan Xiao [aut, cre],
Tengfei Yin [aut],
Emile Young [ctb],
Dusan Randjelovic [ctb],
Seven Bridges Genomics [cph, fnd]

R topics documented:

`sevenbridges-package` *Seven Bridges Platform API Client and CWL Tool Builder in R*

Description

R client and utilities for Seven Bridges platform API, from Cancer Genomics Cloud to other Seven Bridges supported platforms.

Details

Package:	sevenbridges
Type:	Package
License:	Apache License 2.0

Author(s)

Nan Xiao <<nan.xiao@sevenbridges.com>> Tengfei Yin <<tengfei.yin@sevenbridges.com>> Dusan Randjelovic Emile Young

addIdNum *add # prefix to id*

Description

add # prefix to id

Usage

```
addIdNum(x)
```

Arguments

x	(character) with # or not.

Value

a character with # prefix.

Examples

```
addIdNum(c("bam", "#fastq"))
```

api *Core HTTP logic for Seven Bridges API*

Description

Core HTTP logic for Seven Bridges API

Usage

```
api(token = NULL, version = "v2", path = NULL, method = c("GET",
  "POST", "PUT", "DELETE", "PATCH"), query = NULL, body = list(),
  encode = c("json", "form", "multipart"),
  limit = getOption("sevenbridges")$limit,
  offset = getOption("sevenbridges")$offset,
  advance_access = getOption("sevenbridges")$advance_access,
  fields = NULL, base_url = paste0("https://api.sbgenomics.com/",
  version, "/"), ...)
```

Arguments

token	authenticate token string.
version	API version number, default is v2.
path	path connected with base_url.
method	one of "GET", "POST", "PUT", or "Delete".
query	Passed to httr package GET/POST call.
body	Passed to httr package GET/POST/PUT/DELETE 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 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	How many results to return
offset	The point at which to start displaying them
advance_access	Enable advance access features? Default is FALSE.
fields	All API calls take the optional query parameter fields. 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. More details please check https://docs.sevenbridges.com/docs/the-api#section-general-api-information
base_url	defeault is "https://api.sbgenomics.com/v2"
...	passed to GET/POST/PUT/DELETE/PATCH call.

Details

Used for advanced users and the core method for higher level API in this package, please refer to the easy api vignette and additional vignettes pages for more convenient usage.

Value

returned request list of httr

References

https://docs.sevenbridges.com/v1.0/page/api

Examples

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

App-class *Class App*

Description

Class App

Value

App object.

Fields

`id` app id

`project` project id

`name` app name

`revision` app revision

`raw` raw cwl list, if doesn't have any, call `cwl()` method

Examples

```
## Not run:
a <- Auth(url = "https://api.sbgenomics.com/v2/", token = "your_token")
# get a public app
app <- a$public_app(id = "admin/sbg-public-data/rna-seq-alignment-star")
app$input_matrix()
app$output_matrix()
# get a public app
app <- a$public_app(id = "admin/sbg-public-data/star")
app$input_matrix()
app$output_matrix()
## End(Not run)
```

---

asList *Convert a object slots/fields to a list, json, or yaml file*

Description

Doesn't like `as.list`, only fields and slots are converted, prepare a object to be conveted to YAML/JSON.

Usage

```
asList(object, ...)

## S4 method for signature 'ANY'
asList(object, ...)

## S4 method for signature 'CWL'
asList(object, ...)

## S4 method for signature 'SingleEnum'
asList(object, ...)

## S4 method for signature 'SimpleList'
asList(object, ...)

## S4 method for signature 'DSCList'
asList(object, ...)
```

Arguments

`object`	object, could be S4/R5 object. For example, class CWL, SimpleList.
`...`	other parameters passed to `as.yaml` or `toJSON`.

Value

a list object or json or yaml file.

Examples

```
# define a S4 object
A <- setClass("A", slots = list(a = "character", b = "numeric"))
# define a reference object which extends 'CWL' class
B <- setRefClass("B", fields = list(x = "character", y = "A"), contains = "CWL")
# new instances
a <- A(a = "hello", b = 123)
b <- B(x = "world", y = a)

# show
b
b$show("JSON")
b$show("YAML")

# You can convert slots/fields into a list
asList(a)
asList(b)
b$toList()
b$toYAML()
b$toJSON()
```

Auth-class *Class Auth*

Description

Auth object

Details

Every object could be requested from this Auth object and any action could start from this object using cascading style. Please check `vignette("api")` for more information.

Fields

`from` [character] 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` [character] The platform to use. If `platform` and `url` are both not specified, the default is `"cgc"` (Cancer Genomics Cloud). Other possible values include `"aws-us"` (Seven Bridges Platform - US), `"aws-eu"` (Seven Bridges Platform - EU), `"ali-cn"` (Seven Bridges Platform - China), `"cavatica"` (Cavatica), and `"f4c"` (FAIR4CURES).

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

`token` [character] Your authentication token.

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

profile_name [character] Profile name in the user configuration file. The default value is "default".

fs FS object, for mount and unmount file system.

Methods

api(..., limit = getOption("sevenbridges")$limit, offset = getOption("sevenbridges")$offset, fields
: This call returns all API paths, and pass arguments to api() function with input token and url automatically

billing(id = NULL, breakdown = FALSE, ...) If no id provided, This call returns a list of paths used to access billing information via the API. else, This call lists all your billing groups, including groups that are pending or have been disabled. If breakdown = TRUE, This call returns a breakdown of spending per-project for the billing group specified by billing_group. For each project that the billing group is associated with, information is shown on the tasks run, including their initiating user (the runner), start and end times, and cost.

bulk_file_copy(file_ids, project, ...) Copy files between projects in a batch.

bulk_file_delete(file_ids, ...) Delete multiple files.

bulk_file_edit(...) Edit details of multiple files (preserving the omitted fields).

bulk_file_get(file_ids, ...) Get details of multiple files.

bulk_file_update(...) Update details of multiple files (removing the omitted fields).

bulk_task_get(task_ids, ...) Get details of multiple tasks.

bulk_volume_export(...) Bulk export to volumes.

bulk_volume_get_export(...) Get details of a bulk export job.

bulk_volume_get_import(...) Get details of a bulk import job.

bulk_volume_import(...) Bulk import from volumes.

division(id = NULL, ...) List all divisions or get details of a division.

file(name = NULL, id = NULL, project = NULL, exact = FALSE, detail = FALSE, metadata = list(), origin.tas
: This call returns a list of all files in a specified project that you can access. For each file, the call returns: 1) Its ID 2) Its filename The project is specified as a query parameter in the call.

invoice(id = NULL, ...) If no id provided, This call returns a list of invoices, with information about each, including whether or not the invoice is pending and the billing period it covers. The call returns information about all your available invoices, unless you use the query parameter bg_id to specify the ID of a particular billing group, in which case it will return the invoice incurred by that billing group only. If id was provided, This call retrieves information about a selected invoice, including the costs for analysis and storage, and the invoice period.

project(name = NULL, id = NULL, index = NULL, ignore.case = TRUE, exact = FALSE, owner = NULL, detail = FA
: If no id or name provided, this call returns a list of all projects you are a member of. Each project's project_id and URL on the platform will be returned. If name or id provided, we do a match search the list.

project_new(name = NULL, billing_group_id = NULL, description = name, tags = list(), type = "v2", locke
: Create new projects, required parameters: name, billing_group_id, optional parameteres: tags, description, type, and settings.

project_owner(owner = NULL, ...) List the projects owned by and accessible to a particular user. Each project's ID and URL will be returned.

`rate_limit(...)` This call returns information about your current rate limit. This is the number of API calls you can make in one hour.

`send_feedback(text, type = c("idea", "thought", "problem"), referrer = NULL, ...)` Send feedback to Seven Bridges.

`user(username = NULL, ...)` This call returns information about the authenticated user.

`volume(name = NULL, id = NULL, index = NULL, ignore.case = TRUE, exact = FALSE, detail = FALSE, ...)` If no id or name provided, this call returns a list of all volumes you are a member of. If name or id provided, we did a match search the list.

Examples

```
# Direct authentication (default)
# replace with your auth token
token <- "your_token"
a <- Auth(platform = "cgc", token = token)
## Not run:
# Authentication with environment variables
# This will read system environments variables
# `SB_API_ENDPOINT` and `SB_AUTH_TOKEN` by default
a <- Auth(from = "env")

# Authentication with user configuration file
# This will load profile `default` from config
# file `~/.sevenbridges/credentials` by default
a <- Auth(from = "file")

## End(Not run)
```

---

batch *batch function for task batch execution*

Description

batch function for task batch execution

Usage

```
batch(input = NULL, criteria = NULL, type = c("ITEM", "CRITERIA"))
```

Arguments

`input`	character, ID of the input on which you wish to batch on. You would usually batch on the input containing a list of files. If left out, default batching criteria defined in the app is used.
`criteria`	a character vector, for example. `c("metadata.sample_id","metadata.library_id")`. The meaning of the above batch_by dictionary is - group inputs (usually files) first on sample ID and then on library ID. If NULL, using type "ITEM" by default.
`type`	Criteria on which to batch on - can be in two formats."ITEM" and "CRITERIA". If you wish to batch per item in the input (usually a file) using "ITEM". If you wish a more complex criteria, specify the "CRITERIA" on which you wish to group inputs on. Please check examples.

Value

a list of 'batch_input' and 'batch_by' used for task batch

Examples

```
batch(input = "fastq") # by ITEM
batch(input = "fastq", c("metadata.sample_id", "metadata.library_id"))
# shorthand for this
batch(input = "fastq", c("metadata.sample_id", "metadata.library_id"), type = "CRITERIA")
```

Binding-class *Binding*

Description

Binding

Fields

`loadContents` [logical] Only applies when type is File. Read up to the first 64 KiB of text from the file and place it in the "contents" field of the file object for manipulation by expressions.

`secondaryFiles` [] Only applies when type is File. Describes files that must be included alongside the primary file. If the value is Expression, the context of the expression is the input or output File parameter to which this binding applies. Where the value is a string, it specifies that the following pattern should be applied to the primary file: If string begins with one or more caret characters, for each caret, remove the last file extension from the path (the last period . and all following characters). If there are no file extensions, the path is unchanged. Append the remainder of the string to the end of the file path.

Examples

```
Binding(loadContents = TRUE, secondaryFiles = "./test.txt")
```

CCBList *characterORCommandLineBindingList Class*

Description

characterORCommandLineBindingList Class

Usage

```
CCBList(...)
```

Arguments

...	element or list of the element.

Value

CCBList

Examples

```
CCBList("-o output.bam")
```

CLB *Shorthand functions for cwl packages constructors*

Description

Shorthand functions for cwl packages constructors

Arguments

type	[ANY] Specify valid types of data that may be assigned to this parameter.
label	[character] A short, human-readable label of this parameter object.
description	[character] A long, human-readable description of this parameter object.
streamable	[logical] Currently only applies if type is File. A value of true indicates that the file is read or written sequentially without seeking. An implementation may use this flag to indicate whether it is valid to stream file contents using a named pipe. Default: false.
default	[ANY] The default value for this parameter if not provided in the input object.
...	For InputParameter, it will be passed to [CommandLineBinding], which could be created by command CLB. For parameters that accepted please check CommandLineBiding in cwl package. For your convenience, this manual also contain a section for CommandLineBinding. For OutPar or OutputParameter, it will be passed to CommandOutputParameter. Please check the following section as well.

Shorthand

CLB <- CommandLineBinding argslist <- CLBList <- CommandLineBindingList COB <- CommandOutputBinding IPList <- InputParameterList OPList <- OutputParameterList InPar <- InputParameter OutPar <- OutputParameter

CommandLineBinding

position [integer] The sorting key. Default position is 0.

prefix [character] Command line prefix to add before the value.

separate [logical] If true (default) then the prefix and value must be added as separate command line arguments; if false, prefix and value must be concatenated into a single command line argument.

itemSeparator [character] Join the array elements into a single string with the elements separated by by itemSeparator.

valueFrom [characterOrExpression] If valueFrom is a constant string value, use this as the value and apply the binding rules above. If valueFrom is an expression, evaluate the expression to yield the actual value to use to build the command line and apply the binding rules above. If the inputBinding is associated with an input parameter, the "context" of the expression will be the value of the input parameter. When a binding is part of the CommandLineTool.arguments field, the valueFrom field is required.

CommandOutputParameter

glob [characterORExpression] Find files relative to the output directory, using POSIX glob(3) pathname matching. If provided an array, match all patterns in the array. If provided an expression, the expression must return a string or an array of strings, which will then be evaluated as a glob pattern. Only files which actually exist will be matched and returned.

outputEval [Expression] Evaluate an expression to generate the output value. If glob was specified, the script context will be an array containing any files that were matched. Additionally, if loadContents is true, the file objects will include up to the first 64 KiB of file contents in the contents field. Following fields inherited from `Binding`

loadContents [logical] Only applies when type is File. Read up to the first 64 KiB of text from the file and place it in the "contents" field of the file object for manipulation by expressions.

secondaryFiles Only applies when type is File. Describes files that must be included alongside the primary file. If the value is Expression, the context of the expression is the input or output File parameter to which this binding applies. Where the value is a string, it specifies that the following pattern should be applied to the primary file: If string begins with one or more caret characters, for each caret, remove the last file extension from the path (the last period . and all following characters). If there are no file extensions, the path is unchanged. Append the remainder of the string to the end of the file path.

Examples

```
ipl <- IPList(
  input(
    id = "bam",
    type = "File",
    label = "Bam file",
    description = "Input bam file",
    position = 1L,
    separate = TRUE
  ),
  input(
    id = "level",
    type = "Integer",
    label = "Compression Level",
    description = "Set compression level, from 0 (uncompressed) to 9 (best)",
    position = 2L
  ),
  input(
    id = "prefix",
    type = "String",
    label = "Prefix",
    description = "Write temporary files to PREFIX.nnnn.bam",
    position = 3L
  )
)
```

---

cli_list_projects *List projects using Seven Bridges command line uploader*

Description

List projects available as upload targets using Seven Bridges command line uploader.

Usage

```
cli_list_projects(token = NULL, uploader = NULL, proxy = NULL)
```

Arguments

token	Authentication token.
uploader	The directory where Seven Bridges command line uploader is located (the directory that contains the bin/ directory).
proxy	A proxy server through which the uploader should connect. For details the proxy parameter format, see the part on parameter --proxy in the reference below.

Value

Character vector of the available project names.

References

https://docs.sevenbridges.com/docs/upload-via-the-command-line

See Also

See cli_upload for uploading files with the command line uploader, cli_list_tags for listing all tags in a project.

Examples

```
token <- "your_token"
## Not run:
cli_list_projects(
  token = token,
  uploader = "~/Downloads/sbg-uploader/"
)
## End(Not run)
```

---

cli_list_tags *List all the tags in project using Seven Bridges command line uploader*

Description

List all the tags in a destination project using Seven Bridges command line uploader.

Usage

```
cli_list_tags(token = NULL, uploader = NULL, project = NULL,
  proxy = NULL)
```

Arguments

token	Authentication token.
uploader	The directory where Seven Bridges command line uploader is located (the directory that contains the bin/ directory).
project	Unique identifier of the project, for example, "username/project-name".
proxy	A proxy server through which the uploader should connect. For details the proxy parameter format, see the part on parameter --proxy in the reference below.

Value

Chracter vector of file tags in the project.

References

https://docs.sevenbridges.com/docs/upload-via-the-command-line

See Also

See cli_upload for uploading files with the command line uploader, cli_list_projects for listing available projects.

Examples

```
token <- "your_token"
## Not run:
cli_list_tags(
  token = token,
  uploader = "~/Downloads/sbg-uploader/",
  project = "username/project-name"
)
## End(Not run)
```

---

cli_upload *Upload files using Seven Bridges command line uploader*

Description

Upload files using Seven Bridges command line uploader.

Usage

```
cli_upload(token = NULL, uploader = NULL, file = NULL,
  project = NULL, proxy = NULL, tag = NULL, manifest_file = NULL,
  manifest_metadata = c("all", "none", "partial"),
  metadata_fields = NULL, dry_run = FALSE, dry_run_fields = NULL)

misc_upload_cli()
```

Arguments

token	Authentication token.
uploader	The directory where the command line uploader is located (the directory that contains the bin/ directory).
file	The location of the (single) file to upload. To upload multiple files, please use manifest_file to specify.
project	The project identifier (e.g. username/project-name) to upload files to. This option is mandatory. To upload files to a project, you must be a member of that project and must have the write permission granted by the project administrator.
proxy	A proxy server through which the uploader should connect. For details the proxy parameter format, see the part on parameter --proxy in the reference below.
tag	Tags for your the files (optional). Use a vector of character strings, for instance, c("tag one","the second tag").
manifest_file	Location of the manifest file (for uploading multiple files with metadata). See the reference URL below for the format of a manifest file.
manifest_metadata	Should we use all, none, or only a part of the the metadata fields included in the manifest file? Default is "all".
metadata_fields	Character vector, the metadata fields to use in the manifest file. This should be specified if and only if manifest_metadata = "partial".
dry_run	Should we just output the data and check the settings without uploading anything? Default is FALSE.
dry_run_fields	Character vector, specific metadata fields to output information about when dry_run = TRUE.

Value

The uploaded file's ID number.

Note

To use the command line uploader, Java 1.7 or newer should be installed. See the reference link below for details.

References

Seven Bridges Command Line Uploader: https://docs.sevenbridges.com/docs/upload-via-the-command-line
Manifest file format: https://docs.sevenbridges.com/docs/format-of-a-manifest-file

See Also

See get_uploader for downloading the command line uploader for Seven Bridges platforms. See cli_list_projects and cli_list_tags for listing available projects or tags with the command line uploader.

Examples

```
token <- "your_token"
## Not run:
cli_upload(
  token = token,
  uploader = "~/Downloads/cgc-uploader/",
  file = "~/example.fastq", project = "username/project-name"
)
## End(Not run)
```

CommandInputParameter-class

CommandInputParameter Class

Description

An input parameter for a CommandLineTool.

Examples

```
ipl <- InputParameterList(
  CommandInputParameter(
    id = "BAM", type = "File",
    label = "input bam",
    description = "input bam",
    inputBinding = CommandLineBinding(
      position = 1L
    )
  ),
  CommandInputParameter(
    id = "level", type = "Integer",
    label = "Compression level",
    description = "Compression level",
    inputBinding = CommandLineBinding(
      position = 2L,
      prefix = "-l"
    )
  )
)
```

CommandInputSchema-class

CommandInputSchema Class

Description

CommandInputSchema Class

Examples

```
CommandInputSchema()
```

`CommandLineBinding-class` *CommandLineBinding Class*

Description

When listed under inputBinding in the input schema, the term "value" refers to the the corresponding value in the input object. For binding objects listed in CommandLineTool.arguments, the term "value" refers to the effective value after evaluating valueFrom.

Details

The binding behavior when building the command line depends on the data type of the value. If there is a mismatch between the type described by the input schema and the effective value, such as resulting from an expression evaluation, an implementation must use the data type of the effective value.

- characterAdd prefix and the string to the command line.
- numericAdd prefix and decimal representation to command line.
- logicalIf true, add prefix to the command line. If false, add nothing.
- FileAdd prefix and the value of File.path to the command line.
- *ArrayIf itemSeparator is specified, add prefix and the join the array into a single string with itemSeparator separating the items. Otherwise add prefix and recursively add individual elements.
- *objectAdd prefix only, and recursively add object fields for which inputBinding is specified.
- nullAdd nothing.

Fields

`position` [integer] The sorting key. Default position is 0.

`prefix` [character] Command line prefix to add before the value.

`separate` [logical] If true (default) then the prefix and value must be added as separate command line arguments; if false, prefix and value must be concatenated into a single command line argument.

`itemSeparator` [character] Join the array elements into a single string with the elements separated by by itemSeparator.

`valueFrom` [characterOrExpression] If valueFrom is a constant string value, use this as the value and apply the binding rules above. If valueFrom is an expression, evaluate the expression to yield the actual value to use to build the command line and apply the binding rules above. If the inputBinding is associated with an input parameter, the "context" of the expression will be the value of the input parameter. When a binding is part of the CommandLineTool.arguments field, the valueFrom field is required.

Examples

```
CommandLineBinding(position = 1L, prefix = "-l")
```

`CommandLineTool-class` *CommandLineTool Class*

Description

A CommandLineTool process is a process implementation for executing a non-interactive application in a POSIX environment. To help accomodate of the enormous variety in syntax and semantics for input, runtime environment, invocation, and output of arbitrary programs, CommandLineTool provides the concept of "input binding" to describe how to translate input parameters to an actual program invocation, and "output binding" to describe how generate output parameters from program output.

Fields

`baseCommand` (character) Specifies the program to execute. If the value is an array, the first element is the program to execute, and subsequent elements are placed at the beginning of the command line in prior to any command line bindings. If the program includes a path separator character it must be an absolute path, otherwise it is an error. If the program does not include a path separator, search the $PATH variable in the runtime environment find the absolute path of the executable.

`arguments` [characterORCommandLineBinding] Command line bindings which are not directly associated with input parameters.

`stdin` [characterORExpression] A path to a file whose contents must be piped into the command's standard input stream.

`stdout` [characterORExpression] Capture the command's standard output stream to a file written to the designated output directory. If stdout is a string, it specifies the file name to use.If stdout is an expression, the expression is evaluated and must return a string with the file name to use to capture stdout. If the return value is not a string, or the resulting path contains illegal characters (such as the path separator /) it is an error.

`successCodes` [integer] Exit codes that indicate the process completed successfully.

`temporaryFailCodes` [integer] Exit codes that indicate the process failed due to a possibly temporary condition, where excuting the process with the same runtime environment and inputs may produce different results.

`permanentFailCodes` [integer] Exit codes that indicate the process failed due to a permanent logic error, where excuting the process with the same runtime environment and same inputs is expected to always fail.

Input binding

The tool command line is built by applying command line bindings to the input object. Bindings are listed either as part of an input parameter using the inputBinding field, or separately using the arguments field of the CommandLineTool.

The algorithm to build the command line is as follows. In this algorithm, the sort key is a list consisting of one or more numeric and string elements. Strings are sorted lexicographically based on UTF-8 encoding.

- Collect CommandLineBinding objects from arguments. Assign a sorting key [position, i] where position is CommandLineBinding.position and the i is the index in the arguments list.

- Collect CommandLineBinding objects from the inputs schema and associate them with values from the input object. Where the input type is a record, array, or map, recursively walk the schema and input object, collecting nested CommandLineBinding objects and associating them with values from the input object.
- Assign a sorting key for each leaf binding object by appending nested position fields together with the array index, or map key of the data at each nesting level. If two bindings have the same position, the tie must be broken using the lexographic ordering of the field or parameter name immediately containing the binding.
- Sort elements using the assigned sorting keys. Numeric entries sort before strings.
- In the sorted order, apply the rules defined in CommandLineBinding to convert bindings to actual command line elements.
- Insert elements from baseCommand at the beginning of the command line.

Runtime environment

All files listed in the input object must be made available in the runtime environment. The implementation may use a shared or distributed file system or transfer files via explicit download. Implementations may choose not to provide access to files not explicitly specified by the input object or process requirements.

Output files produced by tool execution must be written to the designated output directory.

The initial current working directory when executing the tool must be the designated output directory.

The TMPDIR environment variable must be set in the runtime environment to the designated temporary directory. Any files written to the designated temporary directory may be deleted by the workflow platform when the tool invocation is complete.

An implementation may forbid the tool from writing to any location in the runtime environment file system other than the designated temporary directory and designated output directory. An implementation may provide read-only input files, and disallow in-place update of input files.

The standard input stream and standard output stream may be redirected as described in the stdin and stdout fields.

Extensions

DockerRequirement, CreateFileRequirement, and EnvVarRequirement, are available as standard extensions to core command line tool semantics for defining the runtime environment.

Execution

Once the command line is built and the runtime environment is created, the actual tool is executed.

The standard error stream and standard output stream (unless redirected by setting stdout) may be captured by platform logging facilities for storage and reporting.

Tools may be multithreaded or spawn child processes; however, when the parent process exits, the tool is considered finished regardless of whether any detached child processes are still running. Tools must not require any kind of console, GUI, or web based user interaction in order to start and run to completion.

The exit code of the process indicates if the process completed successfully. By convention, an exit code of zero is treated as success and non-zero exit codes are treated as failure. This may be customized by providing the fields successCodes, temporaryFailCodes, and permanentFailCodes. An implementation may choose to default unspecified non-zero exit codes to either temporaryFailure or permanentFailure.

Output binding

If the output directory contains a file called "cwl.output.json", that file must be loaded and used as the output object. Otherwise, the output object must be generated by walking the parameters listed in outputs and applying output bindings to the tool output. Output bindings are associated with output parameters using the outputBinding field. See CommandOutputBinding for details.

Examples

```
ipl <- InputParameterList(
  InputParameter(
    id = "BAM", type = "File",
    label = "input bam",
    description = "input bam",
    inputBinding = CommandLineBinding(
      position = 1L
    )
  ),
  InputParameter(
    id = "level", type = "Integer",
    label = "Compression level",
    description = "Compression level",
    inputBinding = CommandLineBinding(
      position = 2L,
      prefix = "-l"
    )
  )
)

clt <- CommandLineTool(inputs = ipl, baseCommand = "samtools sort")
```

CommandOutputBinding-class

CommandOutputBinding Class

Description

Describes how to generate an output parameter based on the files produced by a CommandLineTool. The output parameter is generated by applying these operations in the following order: glob, loadContents, outputEval.

Fields

`glob` [characterORExpression] Find files relative to the output directory, using POSIX glob(3) pathname matching. If provided an array, match all patterns in the array. If provided an expression, the expression must return a string or an array of strings, which will then be evaluated as a glob pattern. Only files which actually exist will be matched and returned.

`outputEval` [Expression] Evaluate an expression to generate the output value. If glob was specified, the script context will be an array containing any files that were matched. Additionally, if loadContents is true, the file objects will include up to the first 64 KiB of file contents in the contents field.

Examples

```
CommandOutputBinding(glob = "*.bam")
```

`CommandOutputParameter-class`
CommandOutputParameter Class

Description

CommandOutputParameter Class

Fields

`outputBinding` [CommandOutputBinding] Describes how to handle the concrete outputs of a process step (such as files created by a program) and describe them in the process output parameter.

Examples

```
CommandOutputParameter(outputBinding = CommandOutputBinding(glob = "*.bam"))
```

`CommandOutputSchema-class`
CommandOutputSchema

Description

CommandOutputSchema

Fields

`outputBinding` [CommandOutputBinding] Describes how to handle the concrete outputs of a process step (such as files created by a program) and describe them in the process output parameter.

Examples

```
CommandOutputSchema()
```

`convert_app` *Convert App or a CWL JSON file to Tool or Flow object*

Description

Convert App or a CWL JSON file to Tool or Flow object

Usage

```
convert_app(from)

appType(x)
```

Arguments

`from`	an App object or a CWL JSON
`x`	a App object

Details

This function import CWL JSON file, based on its class: CommandLineTool or Worklfow to relevant object in R, Tool object or Flow object.

Value

Tool or Flow object depends on CWL type.

appType

this function return class of a App object.

Examples

```
tool.in <- system.file("extdata/app", "tool_star.json", package = "sevenbridges")
flow.in <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
# convert to Tool object
convert_app(tool.in)
# convert to Flow object
convert_app(flow.in)
```

`CPURequirement-class` *Rabix specifc Requirements*

Description

Extends ProcessRequirements. CPURequirement and MemRequirement to setup CPU and Memory requiremnts.

requirements and hints

Usage

```
docker(pull = NULL, imageId = NULL, load = NULL, file = NULL,
  output = NULL, dockerPull = pull, dockerImageId = imageId,
  dockerLoad = load, dockerFile = file,
  dockerOutputDirectory = output, ...)

requirements(...)

fileDef(name = NULL, content = NULL)
```

Arguments

pull	[short form argument] Docker Repository[:Tag] like rocker/r-base
imageId	[short form argument] The image id that will be used for docker run, imageId Optionally set the id of image you get from SDK.
load	[short form argument] Specify a HTTP URL from which to download a Docker image using docker load.
file	[short form argument] Supply the contents of a Dockerfile which will be built using docker build.
output	[short form argument] Set the designated output directory to a specific location inside the Docker container.
dockerPull	Docker Repository[:Tag] like rocker/r-base
dockerImageId	The image id that will be used for docker run, imageId Optionally set the id of image you get from SDK.
dockerLoad	Specify a HTTP URL from which to download a Docker image using docker load.
dockerFile	Supply the contents of a Dockerfile which will be built using docker build.
dockerOutputDirectory	Set the designated output directory to a specific location inside the Docker container.
...	extra aguments passed
name	file name
content	file content, could be script

Details

It constructs ProesssRequirementList object, or from a returned raw list contains or requirements.

Value

A Requirement subclass.

Fields

value [Integer] for CPU default is 1L, if 0L, use all CPU. For mem, default is 1000L. Note: for CPU, 0L means multi-tread, and non-zero value will be converted to 1L, which means single thread.

Examples

```
cpu(1)
CPURequirement(value = 1L)
docker("rocker/r-base")
requirements(docker("rocker/r-base"), cpu(1), mem(1024))
mem(2000)
MemRequirement(value = 2000L)
aws("c3.8xlarge")
anyReq("any")
```

CWL-class *Class CWL*

Description

Define CWL class and generic methods, no fields defeind.

Methods

`getFields(values)` Return fields as a list, used for following conversion, does not assume the value is a primitive type.

`toJSON(...)` Covert object to JSON

`toList(...)` Convert object to a list of simple data types

`toYAML(...)` Covert object to YAML

Examples

```
# no fields, only to provide methods to be extended
x <- CWL()
```

---

delete *Delete files or folders*

Description

Delete files or folders

Usage

```
delete(obj)

## S4 method for signature 'SimpleList'
delete(obj)

## S4 method for signature 'Files'
delete(obj)

## S4 method for signature 'Task'
delete(obj)
```

Arguments

obj	single File or FileList

Value

system message

Examples

```
## Not run:
a$project("demo")$file("omni")$delete()
# or
delete(a$project("demo")$file("omni"))
## End(Not run)
```

---

download *Download files*

Description

Download files

Usage

```
download(obj, ...)

## S4 method for signature 'FilesList'
download(obj, ...)

## S4 method for signature 'Files'
download(obj, ...)
```

Arguments

obj	single File or FileList
...	passed to download()

Value

system message

Examples

```
## Not run:
a$project("demo")$file("omni")$download()
# or
download(a$project("demo")$file("omni"))
## End(Not run)
```

DSCList *DSC list*

Description

Contains DataypeSingleEnum, Schema, character

Usage

```
DSCList(...)
```

Arguments

...	element or list of the element.

Value

a DSCList

Examples

```
DSCList("test", DatatypeEnum(), Schema())
```

Expression-class *Expression Class*

Description

Define an expression that will be evaluated and used to modify the behavior of a tool or workflow. See Expressions for more information about expressions and ExpressionEngineRequirement for information on how to define a expression engine.

Fields

`engine` (JsonPointerORcharacter) Either cwl:JsonPointer or a reference to an ExpressionEngineRequirement defining which engine to use.

`script` (character) The code to be executed by the expression engine.

Examples

```
Expression(engine = "#cwl-js-engine", script = "$job.inputs['threads']")
```

ExpressionTool-class *ExpressionTool Class*

Description

Execute an expression as a process step.

Fields

`expression` (Expression) The expression to execute. The expression must return a JSON object which matches the output parameters of the ExpressionTool.

Examples

```
ExpressionTool(
  expression =
    Expression(
      engine = "cwl:JsonPointer",
      script = "$job.inputs['threads']"
    )
)
```

FileList *FileList Class*

Description

FileList Class

File Class

Usage

```
FileList(...)
```

Arguments

`...`	element or list of the element.

Value

File class generator

Fields

`class` (character) Must be File to indicate this object describes a file.

`path` (character) The path to the file.

`checksum` [character] Optional hash code for validating file integrity. Currently must be in the form "sha1$ + hexidecimal string" using the SHA-1 algorithm.

`size` [numeric] Optional file size.

`secondaryFile` [FileList] A list of additional files that are associated with the primary file and must be transferred alongside the primary file. Examples include indexes of the primary file, or external references which must be included when loading primary document. A file object listed in secondaryFiles may itself include secondaryFiles for which the same rules apply.

Examples

```
library(jsonlite)
library(yaml)
f1 <- File()
f2 <- File(path = "./out.bam", checksum = "test",
           size = 3L, secondaryFile = FileList(File(path = "./out.bai")))
fl <- FileList(f1, f2)
asList(fl)
f1
f2
fl
```

`Files-class` *Class Files*

Description

Class Files

Usage

```
FilesList(...)
```

Arguments

`...`	one or more `Files` objects

Details

`Files` (with "s") class is usally returned by the API call which returns Files. A group of `Files` is defined as `FilesList`. Users do not usually need to construct `Files` or `FilesList` manually, they are generated from a API call most of the time.

Value

Files object

Fields

`id` character used as file id

`name` string used as file name

`size` file size

`project` project id if any, when returned by a API call, it usually return the project id and stored with the object.

`created_on` date created on

`modified_on` date modified on

`storage` list as storage type

`origin` list as origin

`tags` list as tags

`metadata` a list for metadata associated with the file

`url` file download url

`parent` parent folder ID

`type` "FILE" or "FOLDER"

`description` file description

Methods

`add_tag(x, ...)` add new tags while keeping old tags

`copy_to(project = NULL, name = NULL)` copy a file to a project (id) with new name

`copy_to_folder(folder_id, name_new = NULL, ...)` Copy a file to a folder.

`create_folder(name, ...)` Create a new folder under the parent folder.

`create_marker(name = NULL, start = NULL, end = NULL, chromosome = NULL, private = TRUE, ...)` Create a marker.

`download(destfile, ..., method = "curl")` see 'help(download.file)' for more options

`get_parent_folder()` Get the parent folder object of the current file/folder.

`get_parent_folder_id()` Get the parent folder ID of the current file/folder.

`list_folder_contents(type = c("file", "folder"), ...)` List folder contents (return files, folders, or both).

`marker(id = NULL, ...)` List markers available on a file or get details for a marker.

`meta()` get metadata from a file

`move_to_folder(folder_id, name_new = NULL, ...)` Move a file to a folder.

`set_meta(..., overwrite = FALSE)` Set metadata with provided list, when overwrite is set to TRUE, it overwrites the metadata.

`set_tag(x = NULL, overwrite = TRUE, ...)` set a tag for a file, your tag need to be a list or vector

`setMeta(..., overwrite = FALSE)` Set metadata with provided list, when overwrite is set to TRUE, it overwrites the metadata.

`tag()` get tag from a file

`typeof()` Get object type ("file" or "folder").

`update(name = NULL, metadata = NULL, tags = NULL)` This call updates the name, the full set metadata, and tags for a specified file.

Note

In the sevenbridges package version <= 1.5.4, the `Files` class inherited from the `File` class defined in CWL. To avoid confusion, in the current implementation, they are defined separately and not coupled anymore.

Examples

```
Files(id = "test_id", name = "test.bam")
```

FS-class *FS class*

Description

FS class

Arguments

server_address	placehoder
api_address	placehoder
vsfs_jar	placehoder
cache_dir	placehoder
cache_size	placeholder
project_id	placeholder

Methods

`file(id = NULL)` given project id, show all files in it

`mount(mount_point = NULL, project_id = NULL, ignore.stdout = TRUE, sudo = TRUE, ...)` mount a specific project if project_id is provided, otherwise mount all projects

`path(id = NULL)` List path for all mounted projects, for easy copy/paste of file path. If project id is provoded, show project path and files path.

`unmount(mount_cmd = NULL, project_id = NULL, ...)` unmount a project if project_id is provided, otherwise unmount all

get_cwl_class *Get class from CWL JSON file*

Description

Get class from CWL JSON file

Usage

```
get_cwl_class(input)
```

Arguments

input	cwl json file path

Value

character for cwl class "Workflow" or "CommandLineTool"

Examples

```
tool.in <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
flow.in <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
get_cwl_class(tool.in)
is_commandlinetool(tool.in)
is_workflow(tool.in)
get_cwl_class(flow.in)
is_commandlinetool(flow.in)
is_workflow(flow.in)
```

---

get_token *Opens web browser to copy the auth token*

Description

Click the "Generate Token" or "Regenerate" button, copy and paste the authentication token string to the R console. The function will return the token string.

Usage

```
get_token(platform = c("cgc", "aws-us", "aws-eu", "gcp", "cavatica"))

misc_get_token()
```

Arguments

platform	The Seven Bridges platform to use.

Value

auth token

Examples

```
token <- NULL
# Will be prompted to enter the auth token
## Not run: token = get_token(platform = "cgc")
```

---

get_uploader *Download Seven Bridges command line uploader and extract to a specified directory*

Description

This function downloads Seven Bridges command line uploader and extract the `.tgz` archive to a specified directory.

Usage

```
get_uploader(platform = c("cgc", "aws-us", "aws-eu", "gcp"),
  destdir = NULL, quiet = FALSE)

misc_get_uploader()
```

Arguments

platform	Seven Bridges platform for which the uploader is designed. Possible choices are: "cgc" (Cancer Genomics Cloud), "aws-us" (Amazon Web Services US), "aws-eu" (Amazon Web Services EU), and "gcp" (Google Cloud Platform). Default is "cgc".
destdir	The directory to extract the downloaded Seven Bridges command line uploader to. If the specified directory is not present, it will be created.
quiet	Should the download progress be printed?

Value

0 if the command line uploader is successfully downloaded and unarchived.

References

https://docs.sevenbridges.com/docs/upload-via-the-command-line

Examples

```
# Download CGC CLI uploader to `~/Downloads`
dir <- "~/Downloads/"
## Not run:
get_uploader("cgc", dir)
## End(Not run)
```

Handler-class *Handler instance*

Description

Create Handler instance

Details

Used for parse R Markdown and lift into command line interface, Dockerfile, Docker container, and cwl json.

Value

a Handler object

Fields

`dockerfileHandler` a function or NULL, how you handle Dockefile, for example, push it to GitHub.

`dockerHandler` a function or NULL, how you handle local docker container, for example, push it to DockerHub.

`cwlHandler` a function or NULL, how you handle cwl json file or yaml file, for example, push it to SevenBridges platform as an app.

---

input_matrix *Get input/output matrix out of JSON CWL file directly*

Description

An efficient way to access JSON file, no need to convert a JSON into a `Tool` or `Flow` object before access, directly operate on a list parsed from JSON file. Compare to `convert_app`, it is much faster.

Usage

```
input_matrix(from, new.order = c("id", "label", "type", "required",
  "prefix", "fileTypes"), required = NULL)

output_matrix(from, new.order = c("id", "label", "type", "fileTypes"))
```

Arguments

`from`	JSON file path
`new.order`	a vector of column orders by default for input it's `"id"`, `"label"`, `"type"`, `"required"`, `"prefix"`, `"fileTypes"`; For output it's `"id"`, `"label"`, `"type"`, `"fileTypes"`
`required`	logical value, show requried input node only or not.

Value

A data frame of input/output information.

Examples

```
tool.in <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
flow.in <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
input_matrix(tool.in)
input_matrix(tool.in, required = TRUE)
input_matrix(flow.in)
input_matrix(flow.in, c("id", "type"))
input_matrix(flow.in, required = TRUE)
tool.in <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
flow.in <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
output_matrix(tool.in)
output_matrix(flow.in)
```

Item-class *Class Item*

Description

Class Item

Details

Base class for describing a set of objects: Project, Task, Pipeline, Files, etc.

Fields

`response` save the raw response from a request.

`auth_token` propagate the auth_token from parent.

`href` API href

link *link two nodes to form a new Workflow*

Description

link two nodes to form a new Workflow

Usage

```
link(from, to, ...)

## S4 method for signature 'Tool,Tool'
link(from, to, id1, id2, flow_id = NULL,
  flow_label = NULL, flow_input = NULL, flow_output = NULL)

## S4 method for signature 'Tool,Workflow'
link(from, to, id1, id2, flow_id = NULL,
  flow_label = NULL, flow_input = NULL, flow_output = NULL)

## S4 method for signature 'Workflow,Tool'
link(from, to, id1, id2, flow_id = NULL,
  flow_label = NULL, flow_input = NULL, flow_output = NULL)

## S4 method for signature 'Workflow,Workflow'
link(from, to, id1, id2)

## S4 method for signature 'App,ToolORWorkflow'
link(from, to, id1, id2)

## S4 method for signature 'ToolORWorkflow,App'
link(from, to, id1, id2)
```

Arguments

from	either Tool App or Workflow object
to	either Tool App or Workflow object
...	more auguments
id1	id to be connected from the ouput of the first node
id2	id id to be connected from the input of the second first node
flow_id	workflow id, if ignored, going to create one by joning tool id.
flow_label	workflow label, if ignored, going to create one by joning tool labels.
flow_input	full flow input id, e.g. "#SBG_Unpack_FASTQs.input_archive_file"
flow_output	full flow output id, e.g. "#STAR.log_files"

Details

Flexible enought to allow users to connect two objects by ids

Value

A Workflow object

Examples

```
t1 <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
t2 <- system.file("extdata/app", "tool_star.json", package = "sevenbridges")
t1 <- convert_app(t1)
t2 <- convert_app(t2)
# check possible link
link_what(t1, t2)
# link
f1 <- link(t1, t2, "output_fastq_files", "reads")
# link
f2 <- link(
  t1, t2, "output_fastq_files", "reads",
  flow_input = "#SBG_Unpack_FASTQs.input_archive_file",
  flow_output = "#STAR.log_files"
)
```

---

link_what *List possible linking methods*

Description

List possible linking methods

Usage

```
link_what(from, to, ...)

## S4 method for signature 'Tool,Tool'
link_what(from, to)

## S4 method for signature 'Tool,SBGWorkflow'
link_what(from, to)

## S4 method for signature 'SBGWorkflow,Tool'
link_what(from, to)

## S4 method for signature 'SBGWorkflow,SBGWorkflow'
link_what(from, to)
```

Arguments

from	either Tool App or SBGWorkflow object
to	either Tool App or Workflow object
...	more auguments

Details

Given two object of Tool, Flow or App, list all possible input/output match.

Value

A Workflow object

Examples

```
t1 <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
t2 <- system.file("extdata/app", "tool_star.json", package = "sevenbridges")
t1 <- convert_app(t1)
t2 <- convert_app(t2)
# check possible link
link_what(t1, t2)
tool.in <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
flow.in <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
t1 <- convert_app(tool.in)
f2 <- convert_app(flow.in)
link_what(t1, f2)
tool.in <- system.file("extdata/app", "tool_unpack_fastq.json", package = "sevenbridges")
flow.in <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
t1 <- convert_app(tool.in)
f2 <- convert_app(flow.in)
link_what(f2, t1)
```

misc_make_metadata *Meta schema*

Description

Meta schema

Usage

```
misc_make_metadata()
```

Details

V2 version for meta data schema

Value

a Metadata object

Examples

```
# show schema (you can still provide customized one)
# empty beause they are all NULL
Metadata()
# show schema
Metadata()$show(TRUE)
# or
names(Metadata()$asList(TRUE))
# returned meta field is actually define as function too, direclty
# call them will give you details
platform()
paired_end()
quality_scale()
# check their suggested value and construct your metadata
Metadata(platform  = "Affymetrix SNP Array 6.0", paired_end = 1, quality_scale = "sanger")
```

Parameter-class *Paramter class (reference class)*

Description

Define an input or output parameter to a process.

Usage

```
InputParameterList(...)

OutputParameterList(...)
```

Arguments

...	element or list of the element.

Value

Parameter object

Fields

`type` [ANY] Specify valid types of data that may be assigned to this parameter.

`label` [character] A short, human-readable label of this parameter object.

`description` [character] A long, human-readable description of this parameter object.

`streamable` [logical] Currently only applies if type is File. A value of true indicates that the file is read or written sequentially without seeking. An implementation may use this flag to indicate whether it is valid to stream file contents using a named pipe. Default: false.

`default` [ANY] The default value for this parameter if not provided in the input object.

`id` (character) The unique identifier for this parameter object.

`inputBinding` [Binding] Describes how to handle the inputs of a process and convert them into a concrete form for execution, such as command line parameters.

`id` (character) The unique identifier for this parameter object.

Examples

```
Parameter(
  type = "integer", label = "thread",
  description = "Specify the thread #",
  default = 0
)

ipl <- InputParameterList(
  InputParameter(
    id = "BAM", type = "File",
    label = "input bam",
    description = "input bam",
    inputBinding = CommandLineBinding(
      position = 1L
    )
  ),
  InputParameter(
    id = "level", type = "Integer",
    label = "Compression level",
    description = "Compression level",
    inputBinding = CommandLineBinding(
      position = 2L,
      prefix = "-l"
    )
  )
)
ipl
```

PrimitiveSingleEnum-class *Pre-defiend enums*

Description

Please check `cwl:::.CWL.Pritimive`, `cwl:::.CWL.Complex`.

Examples

```
PrimitiveEnum()
PrimitiveEnum("boolean")
ComplexEnum("record")
DatatypeEnum("map")
```

Process-class *Process Class*

Description

The base executable type in CWL is the Process object defined by the document. Note that the Process object is abstract and cannot be directly executed.

Fields

`id` [character] The unique identifier for this process object.

`inputs` (InputParameterList) Defines the input parameters of the process. The process is ready to run when all required input parameters are associated with concrete values. Input parameters include a schema for each parameter and is used to validate the input object, it may also be used build a user interface for constructing the input object.

`outputs` (OutputParameterList) Defines the parameters representing the output of the process. May be used to generate and/or validate the output object.

`requirements` [ProcessRequirementList] Declares requirements that apply to either the runtime environment or the workflow engine that must be met in order to execute this process. If an implementation cannot satisfy all requirements, or a requirement is listed which is not recognized by the implementation, it is a fatal error and the implementation must not attempt to run the process, unless overridden at user option.

`hints` [ANY] Declares hints applying to either the runtime environment or the workflow engine that may be helpful in executing this process. It is not an error if an implementation cannot satisfy all hints, however the implementation may report a warning.

`label` [character] A short, human-readable label of this process object.

`description` [character] A long, human-readable description of this process object.

Examples

```
ipl <- InputParameterList(
  InputParameter(
    id = "BAM", type = "File",
    label = "input bam",
    description = "input bam",
    inputBinding = CommandLineBinding(
      position = 1L
    )
  ),
  InputParameter(
    id = "level", type = "Integer",
    label = "Compression level",
    description = "Compression level",
    inputBinding = CommandLineBinding(
      position = 2L,
      prefix = "-l"
    )
  )
)
ipl
p <- Process(id = "process", inputs = ipl)
p
```

ProcessRequirement-class

ProcessRequirement Class

Description

ProcessRequirement Class

DockerRequirement Class

ProcessRequirementList

Usage

```
FileDefList(...)

EnvironmentDefList(...)

ProcessRequirementList(...)
```

Arguments

...	element or list of the element.

Value

a ProcessRequirement object or subclass object.

ProcessRequirement

A process requirement modifies the semantics or runtime environment of a process. If an implementation cannot satisfy all requirements, or a requirement is listed which is not recognized by the implementation, it is a fatal error and the implementation must not attempt to run the process, unless overridden at user option.

(character) The specific requirement type.

DockerRequirement Class

`class` Indicates that a workflow component should be run in a Docker container, and specifies how to fetch or build the image. If a CommandLineTool lists DockerRequirement under hints or requirements, it may (or must) be run in the specified Docker container. The platform must first acquire or install the correct Docker image, as described by DockerRequirement. The platform must execute the tool in the container using docker run with the appropriate Docker image and the tool command line. The workflow platform may provide input files and the designated output directory through the use of volume bind mounts. The platform may rewrite file paths in the input object to correspond to the Docker bind mounted locations. When running a tool contained in Docker, the workflow platform must not assume anything about the contents of the Docker container,such as the presence or absence of specific software, except to assume that the generated command line represents a valid command within the runtime environment of the container.

(character) Get a Docker image using docker pull

`dockerPull` `dockerLoad` (character) Specify a HTTP URL from which to download a Docker image using docker load.

`dockerFile` (character) Supply the contents of a Dockerfile which will be build using docker build.

`dockerImageId` (character) The image id that will be used for docker run. May be a human-readable image name or the image identifier hash. May be skipped if dockerPull is specified, in which case the dockerPull image id will be used.

`dockerOutputDirectory` (character) Set the designated output directory to a specific location inside the Docker container.

SubworkflowFeatureRequirement Class

Indicates that the workflow platform must support nested workflows in the run field of (WorkflowStep)(#workflowstep).

FileDef Class

Define a file that must be placed by in the designated output directory prior to executing the command line tool. May be the result of executing an expression, such as building a configuration file from a template.

(characterORExpression) The name of the file to create in the output directory.

`filename` `fileContent` (characterORExpression) If the value is a string literal or an expression which evalutes to a string, a new file must be created with the string as the file contents. If the value is an expression that evaluates to a File object, this indicates the referenced file should be added to the designated output directory prior to executing the tool. Files added in this way may be read-only, and may be implemented through bind mounts or file system links in such a way as to avoid unecessary copying of the input file.

CreateFileRequirement Class

Define a list of files that must be created and placed by the workflow platform in the designated output directory prior to executing the command line tool. See FileDef for details.

(FileDefList) The list of files.

EnvironmentDef Class

`fileDef` Define an environment variable that will be set in the runtime environment by the workflow platform when executing the command line tool. May be the result of executing an expression, such as getting a parameter from input.

(character) The environment variable name.

`envName` `envValue` (characterORExpression) The environment variable value.

EnvVarRequirement Class

Define a list of environment variables which will be set in the execution environment of the tool. See EnvironmentDef for details.

(EnvironmentDefList) The list of environment variables.

ScatterFeatureRequirement Class

`envDef` Indicates that the workflow platform must support the scatter and scatterMethod fields of (WorkflowStep)(#workflowstep).

ExpressionEngineRequirement Class

Define an expression engine, as described in Expressions.

(character) Used to identify the expression engine in the engine field of Expressions.

`id` `requirements` [ProcessRequirement]Requirements to run this expression engine, such as DockerRequirement for specifying a container with the engine.

`engineCommand` [character] The command line to invoke the expression engine.

`engineConfig` [character] Additional configuration or code fragments that will also be passed to the expression engine. The semantics of this field are defined by the underlying expression engine. Intended for uses such as providing function definitions that will be called from CWL expressions.

Examples

```
dkr <- DockerRequirement(dockerImageId = "testid")
cfr <- CreateFileRequirement(fileDef = FileDefList(FileDef(filename = "hello.txt")))
sfr <- SubworkflowFeatureRequirement()
evr <- EnvVarRequirement(envDef = EnvironmentDefList(
  EnvironmentDef(envName = "path", envValue = "testpath")
))
safr <- ScatterFeatureRequirement()
eer <- ExpressionEngineRequirement(id = "hello")
ProcessRequirementList(dkr, cfr, sfr, evr, safr, eer)
```

project_details *Returns the details of the project*

Description

Returns the details of the project.

Usage

```
project_details(token = NULL, project_id = NULL, ...)
```

Arguments

token	auth token
project_id	ID of a project you want to access.
...	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- project_details(token, project_id = "your_project_id")
## End(Not run)
```

project_members *Returns a list of all users invited to the project and their privileges*

Description

Returns a list of all users invited to the project and their privileges. Project ID is specified as path parameter. Call returns ID and username of the user with privileges.

Usage

```
project_members(token = NULL, project_id = NULL, ...)
```

Arguments

token	auth token
project_id	ID of a project you want to access.
...	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- project_members(token, project_id = "your_project_id")
## End(Not run)
```

`response` *Get raw response from an Item object*

Description

Get raw response from an Item object

Usage

```
response(x)

response(x) <- value

## S4 method for signature 'ANY'
response(x)

## S4 replacement method for signature 'ANY'
response(x) <- value

## S4 method for signature 'Item'
response(x)

## S4 replacement method for signature 'Item'
response(x) <- value

## S4 method for signature 'SimpleList'
response(x)

## S4 replacement method for signature 'SimpleList'
response(x) <- value
```

Arguments

`x`	object that may have response.
`value`	value to be replaced.

Value

a raw response from httr

Examples

```
## Not run:
response(x)
## End(Not run)
```

SBGWorkflow-class *Build workflow*

Description

Build workflow

Usage

```
Flow(..., graph = TRUE, x.width = 1000, y.width = 400,
  x.start = 100, y.start = 200, canvas_zoom = 1, canvas_x = 40,
  canvas_y = 130)

## S4 method for signature 'Tool,Tool'
e1 + e2

## S4 method for signature 'WorkflowStepList,Tool'
e1 + e2

## S4 method for signature 'WorkflowStepList,WorkflowStepList'
e1 + e2

## S4 method for signature 'App,App'
e1 + e2

## S4 method for signature 'WorkflowStepList,App'
e1 + e2

e1 %>>% e2

## S4 method for signature 'Tool,Tool'
e1 %>>% e2

## S4 method for signature 'Workflow,Tool'
e1 %>>% e2

## S4 method for signature 'Workflow,Workflow'
e1 %>>% e2

## S4 method for signature 'App,App'
e1 %>>% e2

## S4 method for signature 'Workflow,App'
e1 %>>% e2
```

Arguments

...	extra arguments passed to SBGWorkflow
graph	if add graph coordinates or not, used for flow visualization on Seven Bridges platforms.

x.width	x scale width
y.width	y scale width
x.start	node x start point for a flow
y.start	node y start point for a flow
canvas_zoom	zoom factor
canvas_x	canvas x
canvas_y	canvas y
e1	either Tool App or Workflow object
e2	either Tool App or Workflow object

Value

a SBGWorkflow object.

Methods

`copy_obj()` this is a hack to make copy of reference cwl object

`get_input(ids, force = FALSE)` get input by pure input id from all steps

`get_input_exposed()` exposed input id other than file

`get_input_node()` get input file nodes id

`get_input_port()` show included port of all inputs

`get_output(ids, force = FALSE)` get output by pure output id from all steps

`get_output_node()` get output file nodes id

`get_required()` show flow required input id and types

`get_step(name = NULL, id = NULL)` get step object by name or id, name support pattern match

`get_tool(name = NULL, id = NULL)` get a tool object by name or id, name support pattern match

`input_id()` show input id

`input_matrix(new.order = c("id", "label", "type", "required", "prefix", "fileTypes"), required = NULL` This return a matrix of input parameters, by default, following the order id, label, type, required, prefix, fileTypes. new.order accept names of column you want to print, but it has to be a field of inputs. When its set to NULL, it prints all fields. When required = TRUE, only print required field.

`input_type()` Show a vector of flow input type, names of them are input id.

`link_map()` show a table of all linked nodes

`linked_input_id()` input id that linked to an output

`linked_output_id()` output id that linked to an input

`list_tool()` list all tools included in this flow

`output_id()` show output id

`output_matrix(new.order = c("id", "label", "type", "fileTypes"))` This return a matrix of output parameters, by default, following the order id, label, type, fileTypes. new.order accept names of column you want to print, but it has to be a field of outputs. When its set to NULL, it prints all fields. When required = TRUE, only print required field.

`output_type()` Show a vector of flow output type, names of them are output id.

`run(run_inputs = list(), engine = c("bunny", "rabix", "cwlrun"))` Run this tool with inputs locally. Engines supported: bunny, rabix, cwlrun. Inputs accept list or JSON.

`set_batch(input = NULL, criteria = NULL, type = c("ITEM", "CRITERIA"))` Set a flow input node into a batch mode, this is now required before you execute a batch task on a batch-not-enabled flow.

`set_flow_input(iid = NULL, add = TRUE)` Expose tool input node as flow input, default is additative, if add = FALSE, this will overwrite and only made provided id inputs of flow.

`set_flow_output(oid = NULL, add = TRUE)` Expose tool output node as flow output, default is additative, if add = FALSE, this will overwrite and only made provided id outputs of flow.

`set_input_port(ids, include = TRUE)` set included port for provided input id(s)

`set_required(ids, required = TRUE)` Set a input node required (TRUE) or not required (FALSE) this require full input id (with tool id prefix) such as #STAR.alignIntronMax

`step_input_id(full = FALSE)` Show step input id, default names of them is tool id. When full = TRUE, show full name then names of vector is type.

`step_output_id(full = FALSE)` Show step output id, default names of them is tool id. when full = TRUE, show full name then names of vector is type.

Examples

```
f1 <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
f1 <- convert_app(f1)
# input matrix
f1$input_matrix()
# by name
f1$input_matrix(c("id", "type", "required", "link_to"))
# return only required
f1$input_matrix(required = TRUE)
# return everything
f1$input_matrix(NULL)
# return a output matrix with more informtion
f1$output_matrix()
# return only a few fields
f1$output_matrix(c("id", "type"))
# return everything
f1$output_matrix(NULL)
# flow inputs
f1$input_type()
# flow outouts
f1$output_type()
# flow input id
f1$input_id()
# linked input id
f1$linked_input_id()
# flow output id
f1$output_id()
# linked output id
f1$linked_output_id()
# link_map
f1$link_map()
# all step input id
f1$step_input_id()
# all step input full id with type
f1$step_input_id(TRUE)
# all step output id
f1$step_output_id()
# all step output full id with type
```

```
f1$step_output_id(TRUE)
# get inputs objects
f1$get_input("#clip3pNbases")
f1$get_input(c("#clip3pNbases", "#chimScoreMin"))
f1$get_input(c("#clip3pNbases", "#chimScoreMin", "#STAR.outFilterMismatchNoverLmax"))
# get outputs objects
f1$get_output("#log_files")
f1$get_output(c("#log_files", "intermediate_genome"))
f1$get_output(c("#log_files", "intermediate_genome", "#STAR.unmapped_reads"))
f1$get_output("#log_files")
# set flow input
f1$set_flow_input("#SBG_FASTQ_Quality_Detector.fastq")
f1$set_flow_output(c("#log_files", "intermediate_genome"))
# get required node
f1$get_required()
# set required node
f1$steps[[1]]$run$set_required("genomeChrBinNbits")
f1$get_required()
f1$steps[[1]]$run$set_required("genomeChrBinNbits", FALSE)
f1$get_required()
# get Tool object from Flow by id and name
f1$list_tool()
# return two
f1$get_tool("STAR")
# return one
f1$get_tool("^STAR$")
# get included input ports
f1$get_input_port()
# set included input ports
f1$set_input_port(c("#STAR.alignSJDBoverhangMin", "chimScoreSeparation"))
f1$get_input_port()
f1$set_input_port(c("#STAR.alignSJDBoverhangMin", "chimScoreSeparation"), FALSE)
f1$get_input_port()
f1$get_input_node()
f1$get_output_node()
f1$get_input_exposed()
f1$step_input_id(TRUE)
f1$input_id()
f1$set_flow_input("#STAR.reads")
f1$input_id()
# batch
f1$set_batch("sjdbGTFfile", c("metadata.sample_id", "metadata.library_id"))
f1$set_batch("sjdbGTFfile", type = "ITEM")
# add source to id
f1$link_map()
f1$add_source_to_id(c("test1", "test2"), c("#STAR.genome", "#STAR.reads"))
f1$link_map()
```

---

sbg_get_env *Set authentication environment variables for Seven Bridges API*

Description

Set authentication environment variables for Seven Bridges API

Usage

```
sbg_get_env(x)
```

Arguments

x	Name of the system environment variable

Value

value of the environment variable

Examples

```
# set and get two environment variables for CGC
token <- "your_token"
## Not run:
sbg_set_env("https://cgc-api.sbgenomics.com/v2", token)
sbg_get_env("SB_API_ENDPOINT")
sbg_get_env("SB_AUTH_TOKEN")
## End(Not run)
```

---

sbg_set_env *Set authentication environment variables for Seven Bridges API*

Description

Set authentication environment variables for Seven Bridges API

Usage

```
sbg_set_env(url = NULL, token = NULL)
```

Arguments

url	Base URL for API.
token	Your authentication token.

Value

set two environment variables for authentication

Examples

```
# set and get environment variables for CGC
token <- "your_token"
## Not run:
sbg_set_env("https://cgc-api.sbgenomics.com/v2", token)
sbg_get_env("SB_API_ENDPOINT")
sbg_get_env("SB_AUTH_TOKEN")
## End(Not run)
```

SchemaList *SchemaList*

Description

A schema defines a parameter type.

Usage

```
SchemaList(...)

SchemaDefList(...)
```

Arguments

...	element or list of the element.

Value

a Schema object or sbuclass object.

Fields

`type` (ANY) The data type of this parameter.

`fields` [SchemaList] When type is record, defines the fields of the record.

`symbols` [character] When type is enum, defines the set of valid symbols.

`items` [ANY] When type is array, defines the type of the array elements.

`values` [ANY] When type is map, defines the value type for the key/value pairs.

`inputBinding` [Binding] Describes how to handle a value in the input object convert it into a concrete form for execution, such as command line parameters.

Examples

```
Schema(fields = SchemaList(SchemaDef(name = "schema")))
```

setListClass *List Class generator.*

Description

Extends IRanges SimpleList class and return constructor.

Usage

```
setListClass(elementType = NULL, suffix = "List", contains = NULL,
  where = topenv(parent.frame()))
```

Arguments

elementType	[character]
suffix	[character] default is "List"
contains	[character] class name.
where	environment.

Value

S4 class constructor

---

setTaskHook *set task function hook*

Description

set task function hook according to

Usage

```
setTaskHook(status = c("queued", "draft", "running", "completed",
  "aborted", "failed"), fun)

getTaskHook(status = c("queued", "draft", "running", "completed",
  "aborted", "failed"))
```

Arguments

status	one of "queued", "draft", "running", "completed", "aborted", or "failed".
fun	function it must return a TRUE or FALSE in the end of function body, when it's TRUE this function will also terminate monitor process, if FALSE, function called, but not going to terminate task monitoring process.

Value

object from setHook and getHook.

Examples

```
getTaskHook("completed")
setTaskHook("completed", function() {
  message("completed")
  return(TRUE)
})
```

set_tag *Set file tags*

Description

Set file tags

Add new file tags and keep the old tags

Usage

```
set_tag(obj, ...)

## S4 method for signature 'FilesList'
set_tag(obj, ...)

## S4 method for signature 'Files'
set_tag(obj, ...)

add_tag(obj, ...)

## S4 method for signature 'FilesList'
add_tag(obj, ...)

## S4 method for signature 'Files'
add_tag(obj, ...)
```

Arguments

obj	single File or FileList
...	passed to obj$set_tag() or obj$add_tag()

Value

tag list

tag list

Examples

```
## Not run:
fl <- a$project("demo")$file("omni")
set_tag(fl, "new tag")
set_tag(fl, list("new tag", "new tag 2"))
## End(Not run)
## Not run:
fl <- a$project("demo")$file("omni")
add_tag(fl, "new tag")
add_tag(fl, list("new tag", "new tag 2"))
## End(Not run)
```

set_test_env *Set testing env*

Description

Checks if docker is installed, is running and has required images downloaded and if do creates container

Usage

```
set_test_env(type = "host", docker_image = "tengfei/testenv",
  data_dir = getwd())
```

Arguments

type	"dind" or "host"
docker_image	required docker image with pre-installed bunny, default: tengfei/testenv
data_dir	directory with data to mount (also will be execution directory)

Value

docker stdout

Examples

```
## Not run:
set_test_env("dind", "tengfei/testenv", "/Users/<user>/tools")
## End(Not run)
```

test_tool_bunny *Test tools in rabix/bunny*

Description

Test tools locally in rabix/bunny inside docker container

Usage

```
test_tool_bunny(rabix_tool, inputs)
```

Arguments

rabix_tool	rabix tool from Tool class
inputs	input parameters declared as json (or yaml) string

Value

bunny stdout

Examples

```
## Not run:
inputs <- '{"counts_file": {"class": "File", "path": "./FPKM.txt"}, "gene_names": "BRCA1"}'
rbx <- <define rabix tool>
set_test_env("tengfei/testenv", "<mount_dir>")
test_tool_bunny(rbx, inputs)
## End(Not run)
```

`test_tool_cwlrun` *Test tools with cwl-runner*

Description

Test tools locally cwl-runner (https://github.com/common-workflow-language/cwltool)

Usage

```
test_tool_cwlrun(rabix_tool, inputs = list())
```

Arguments

`rabix_tool`	rabix tool from Tool class
`inputs`	input parameters declared as json (or yaml) string

Value

cwl-runner stdout

Examples

```
## Not run:
inputs <- '{"counts_file": {"class": "File", "path": "./FPKM.txt"}, "gene_names": "BRCA1"}'
rbx <- <define rabix tool>
set_test_env("tengfei/testenv", "<mount_dir>")
test_tool_cwlrun(rbx, inputs)

## End(Not run)
```

`test_tool_rabix` *Test tools in rabix/rabix-devel (DEPRECATED)*

Description

Test tools locally in rabix/rabix-devel python executor (DEPRECATED)

Usage

```
test_tool_rabix(rabix_tool, inputs = list())
```

Arguments

rabix_tool	rabix tool from Tool class
inputs	input parameters declared as json (or yaml) string

Value

rabix stdout

Examples

```
## Not run:
inputs <- '{"counts_file": {"class": "File", "path": "./FPKM.txt"}, "gene_names": "BRCA1"}'
rbx <- <define rabix tool>
set_test_env("tengfei/testenv", "<mount_dir>")
test_tool_rabix(rbx, inputs)
## End(Not run)
```

Tool-class *Class Tool*

Description

codeTool class extends CommandLineTool with more seven bridges flabored fields the SBG class. obj$toJSON(), obj$toJSON(pretty = TRUE) or obj$toYAML() will convert a Tool object into a text JSON/YAML file.

Value

a Tool object.

Fields

context [character] by default: http://www.commonwl.org/draft-2/

owner [list] a list of owner names.

contributor [list] a list of contributor names.

Methods

copy_obj() this is a hack to make copy of reference cwl object

get_input(name = NULL, id = NULL) get input objects by names or id

get_input_port() the inputs node with sbg:includeInPorts equals TRUE

get_output(name = NULL, id = NULL) get output objects by names or id

get_required() return required input fields types, names of them are input id

input_id(full = FALSE, requiredOnly = FALSE) Get input id from a Tool, when full = TRUE, connect tool id with input id. e.g. If requiredOnly = TRUE, return required field only.

input_matrix(new.order = c("id", "label", "type", "required", "prefix", "fileTypes"), required = NULL This return a matrix of input parameters, by default, following the order id, label, type, required, prefix, fileTypes. new.order accept names of column you want to print, but it has to be a field of inputs. When its set to NULL, it prints all fields. When required = TRUE, only print required field.

`input_type()` this return a vector of types, names of them are input id

`output_id(full = FALSE)` Get output id from a Tool, when full = TRUE, connect tool id with input id.

`output_matrix(new.order = c("id", "label", "type", "fileTypes"))` This return a matrix of output parameters, by default, following the order id, label, type, fileTypes. new.order accept names of column you want to print, but it has to be a field of outputs. When its set to NULL, it prints all fields. when required = TRUE, only print required field.

`output_type()` this return a vector of types, names of them are output id

`run(run_inputs = list(), engine = c("bunny", "rabix", "cwlrun"))` Run this tool with inputs locally. Engines supported: bunny, rabix, cwlrun. Inputs accept list or JSON.

`set_input_port(ids, include = TRUE)` Set inputs ports field sbg:includeInPorts to the value of include, default is TRUE.

`set_required(ids, required = TRUE)` Set an input node required or not required. The first parameter takes single input id or more than one ids. The second parameters required is the value you want to set to inputs. TRUE means set to required.

Examples

```
t1 <- system.file("extdata/app", "tool_star.json", package = "sevenbridges")
# convert json file into a Tool object
t1 <- convert_app(t1)
# get input type information
t1$input_type()
# get output type information
t1$output_type()
# return a input matrix with more informtion
t1$input_matrix()
# return only a few fields
t1$input_matrix(c("id", "type", "required"))
# return only required
t1$input_matrix(required = TRUE)
# return everything
t1$input_matrix(NULL)
# return a output matrix with more informtion
t1$output_matrix()
# return only a few fields
t1$output_matrix(c("id", "type"))
# return everything
t1$output_matrix(NULL)
# get input id
t1$input_id()
# get full input id with Tool name
t1$input_id(TRUE)
# get output id
t1$output_id()
# get full output id
t1$output_id(TRUE)
# get required input id
t1$get_required()
# set input required
t1$set_required(c("#reads", "winFlankNbins"))
t1$get_required()
t1$set_required("reads", FALSE)
t1$get_required()
```

```
t1$get_input(name = "ins")
t1$get_input(id = "#winFlankNbins")
t1$get_output(name = "gene")
t1$get_output(id = "#aligned_reads")
# get a tool from a flow
f1 <- system.file("extdata/app", "flow_star.json", package = "sevenbridges")
# convert json file into a Tool object
f1 <- convert_app(f1)
t2 <- f1$get_tool("STAR$")
oid <- t2$get_input_port()
oid
# set new ports
t2$input_id()
t2$set_input_port("#chimScoreSeparation")
t2$get_input_port()
t2$set_input_port("#chimScoreSeparation", FALSE)
t2$get_input_port()
# run the tool locally with example data
## Not run:
t3 <- system.file("extdata/app/dna2protein", "translate.cwl.json", package = "sevenbridges")
t3 <- convert_app(t3)
fl <- system.file("extdata/app/dna2protein/data", "input.txt", package = "sevenbridges")
set_test_env("dind", "tengfei/testenv", "~/mounts")
t3$input_type()
t3$run(list(input_file = Files(fl)))  # Not File
## End(Not run)
```

`upload_complete_all` *Reports the complete file upload*

Description

If the whole parts are uploaded, and the provided ETags are correct, then the file is assembled and made available on the SBG platform.

Usage

```
upload_complete_all(token = NULL, upload_id = NULL, ...)
```

Arguments

`token`	auth token
`upload_id`	ID of the upload
`...`	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- upload_complete_all(token, upload_id = "your_upload_id")
## End(Not run)
```

upload_complete_part *Reports the completion of the part upload*

Description

The ETag is provided for the correctness check upon completion of the whole upload. Value for the ETag is provided by AWS S3 service when uploading the file in the ETag header.

Usage

```
upload_complete_part(token = NULL, upload_id = NULL,
  part_number = NULL, e_tag = NULL, ...)
```

Arguments

token	auth token
upload_id	ID of the upload
part_number	ID of the part you wish to report as completed
e_tag	Value of the ETag header returned by AWS S3 when uploading part of the file.
...	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- upload_complete_part(
  token, upload_id = "your_upload_id",
  part_number = "1", e_tag = "your_e_tag"
)
## End(Not run)
```

upload_delete *Abort the upload*

Description

Abort the upload; all upload records and the file are deleted.

Usage

```
upload_delete(token = NULL, upload_id = NULL, ...)
```

Arguments

token	auth token
upload_id	ID of the upload
...	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- upload_delete(token, upload_id = "your_upload_id")
## End(Not run)
```

---

upload_info *Returns upload information for the ongoing upload*

Description

Returns the upload information for the ongoing upload.

Usage

```
upload_info(token = NULL, upload_id = NULL, ...)
```

Arguments

token	auth token
upload_id	ID of the upload
...	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run: req <- upload_info(token, upload_id = "your_upload_id")
```

upload_info_part *Returns AWS S3 signed URL for a part of the file upload*

Description

Gets the signed URL for the upload of the specified part. Note that URLs are valid for 60 seconds only and that you should initiate upload to the signed URL in this time frame.

Usage

```
upload_info_part(token = NULL, upload_id = NULL, part_number = NULL,
  ...)
```

Arguments

token	auth token
upload_id	ID of the upload
part_number	Number of the upload file part that you wish to access
...	parameters passed to api function

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- upload_info_part(token, upload_id = "your_upload_id", part_number = 1)
## End(Not run)
```

upload_init *Initializes the upload of the specified file*

Description

This is the first operation performed when you wish to upload a file. Operation is initialized by providing file name, project id where you wish the file to be uploaded to (if not specified, defaults to user's stash) and optionally by providing wanted part size. You may wish to set your part size to a low value if you experience problems with uploading large file parts, although default value of 5MB should be good enough for most users.

Usage

```
upload_init(token = NULL, project_id = NULL, name = NULL,
  size = NULL, part_size = NULL, ...)
```

Arguments

token	auth token
project_id	ID of the project you wish to upload to
name	Name of the file you wish to upload
size	Size of the file you wish to upload
part_size	Requested part size. Note that API may reject your requested part size and return proper one in response.
...	parameters passed to api function

Details

Limits:

- Maximum number of parts is 10000
- Maximum file size is 5TB
- Maximum part size is 5GB
- Default part size is 5MB

Value

parsed list of the returned json

Examples

```
token <- "your_token"
## Not run:
req <- upload_init(
  token,
  project_id = "your_project_id",
  name = "Sample1_RNASeq_chr20.pe_1.fastq", size = 5242880
)
## End(Not run)
```

WorkflowOutputParameter-class

Workflow

Description

A workflow is a process consisting of one or more steps. Each step has input and output parameters defined by the inputs and outputs fields. A workflow executes as described in execution model.

Usage

```
WorkflowOutputParameterList(...)
```

Arguments

...	element or list of the element.

Value

a Workflow object.

Fields

`outputs` (WorkflowOutputParameterList) Defines the parameters representing the output of the process. May be used to generate and/or validate the output object. Inherited from Process

`steps` (WorkflowStepList) The individual steps that make up the workflow. Steps are executed when all input data links are fufilled. An implementation may choose to execute the steps in a different order than listed and/or execute steps concurrently, provided that dependencies between steps are met.

WorkflowOutputParameter Class

Describe an output parameter of a workflow. The parameter must be connected to one or more parameters defined in the workflow that will provide the value of the output parameter.

[character] Specifies one or more workflow parameters that will provide this output value.

`source` `linkMerge` [LinkMergeMethod] The method to use to merge multiple inbound links into a single array. If not specified, the default method is merge_nested:

Dependencies

Dependencies between parameters are expressed using the source field on workflow step input parameters and workflow output parameters.

The source field expresses the dependency of one parameter on another such that when a value is associated with the parameter specified by source, that value is propagated to the destination parameter. When all data links inbound to a given step are fufilled, the step is ready to execute.

Extensions

ScatterFeatureRequirement and SubworkflowFeatureRequirement are available as standard extensions to core workflow semantics.

Examples

```
# need better examples here
ws <- WorkflowStepList(
    WorkflowStep(
        id = "step1", label = "align-and-sort",
        description = "align and sort",
        inputs = WorkflowStepInputList(
            WorkflowStepInput(id = "id1"),
            WorkflowStepInput(id = "id2")
        )
    )
)
Workflow(steps = ws)
```

`WorkflowStepInput-class` *WorkflowStepInputList*

Description

A workflow step is an executable element of a workflow. It specifies the underlying process implementation (such as CommandLineTool) in the run field and connects the input and output parameters of the underlying process to workflow parameters.

Usage

```
WorkflowStepInputList(...)

WorkflowStepOutputList(...)

WorkflowStepList(...)
```

Arguments

`...`	element or list of the element.

Value

a WorkflowStep object or subclass object.

Fields

`id` [character] The unique identifier for this workflow step.

`inputs` (WorkflowStepInputList) Defines the input parameters of the workflow step. The process is ready to run when all required input parameters are associated with concrete values. Input parameters include a schema for each parameter and is used to validate the input object, it may also be used build a user interface for constructing the input object.

`outputs` (WorkflowStepOutputList) Defines the parameters representing the output of the process. May be used to generate and/or validate the output object.

`requirements` [ProcessRequirement] Declares requirements that apply to either the runtime environment or the workflow engine that must be met in order to execute this workflow step. If an implementation cannot satisfy all requirements, or a requirement is listed which is not recognized by the implementation, it is a fatal error and the implementation must not attempt to run the process, unless overridden at user option.

`hints` [ANY] Declares hints applying to either the runtime environment or the workflow engine that may be helpful in executing this workflow step. It is not an error if an implementation cannot satisfy all hints, however the implementation may report a warning.

`label` [character] A short, human-readable label of this process object.

`description` [character] A long, human-readable description of this process object.

`run` (CommandLineToolORExpressionToolORWorkflow) Specifies the process to run.

`scatter` [character]

`scatterMethod` [ScatterMethod] Required if scatter is an array of more than one element.

WorkflowStepInput Class

The input of a workflow step connects an upstream parameter (from the workflow inputs, or the outputs of other workflows steps) with the input parameters of the underlying process.

If the sink parameter is an array, or named in a workflow scatter operation, there may be multiple inbound data links listed in the connect field. The values from the input links are merged depending on the method specified in the linkMerge field. If not specified, the default method is merge_nested:

The input shall be an array consisting of exactly one entry for each input link. If merge_nested is specified with a single link, the value from the link is wrapped in a single-item list.

merge_nested **merge_flattened** 1) The source and sink parameters must be compatible types, or the source type must be compatible with single element from the "items" type of the destination array parameter. 2) Source parameters which are arrays are concatenated; source parameters which are single element types are appended as single elements.

Fields:

`id` (character) A unique identifier for this workflow input parameter.

`source` [character] Specifies one or more workflow parameters that will provide input to the underlying process parameter.

`linkMerge` [LineMergeMethod] The method to use to merge multiple inbound links into a single array. If not specified, the default method is merge_nested:

`default` [ANY] The default value for this parameter if there is no source field.

WorkflowStepOutput Class

Associate an output parameter of the underlying process with a workflow parameter. The workflow parameter (given in the id field) be may be used as a source to connect with input parameters of other workflow steps, or with an output parameter of the process.

(character) A unique identifier for this workflow output parameter. This is the identifier to use in the source field of WorkflowStepInput to connect the output value to downstream parameters.

Scatter/gather

`id` To use scatter/gather, ScatterFeatureRequirement must be specified in the workflow or workflow step requirements.

A "scatter" operation specifies that the associated workflow step or subworkflow should execute separately over a list of input elements. Each job making up a scatter operaution is independent and may be executed concurrently.

The scatter field specifies one or more input parameters which will be scattered. An input parameter may be listed more than once. The declared type of each input parameter is implicitly wrapped in an array for each time it appears in the scatter field. As a result, upstream parameters which are connected to scattered parameters may be arrays.

All output parameters types are also implicitly wrapped in arrays; each job in the scatter results in an entry in the output array.

If scatter declares more than one input parameter, scatterMethod describes how to decompose the input into a discrete set of jobs.

- dotproduct specifies that each the input arrays are aligned and one element taken from each array to construct each job. It is an error if all input arrays are not the same length.

- nested_crossproductspecifies the cartesian product of the inputs, producing a job for every combination of the scattered inputs. The output must be nested arrays for each level of scattering, in the order that the input arrays are listed in the scatter field.
- flat_crossproductspecifies the cartesian product of the inputs, producing a job for every combination of the scattered inputs. The output arrays must be flattened to a single level, but otherwise listed in the order that the input arrays are listed in the scatter field.

Subworkflows

To specify a nested workflow as part of a workflow step, SubworkflowFeatureRequirement must be specified in the workflow or workflow step requirements.

Examples

```
ws <- WorkflowStepList(WorkflowStep(
  id = "step1", label = "align-and-sort",
  description = "align and sort",
  inputs = WorkflowStepInputList(
    WorkflowStepInput(id = "id1"),
    WorkflowStepInput(id = "id2")
  )
))
```

Index