Package ‘biocompute’

November 28, 2019

Type Package

Title Create and Manipulate BioCompute Objects

Version 1.0.3

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

Description Tools to create, validate, and export BioCompute Objects described in King et al. (2019) <doi:10.17605/osf.io/h59uh>. Users can encode information in data frames, and compose BioCompute Objects from the domains defined by the standard. A checksum validator and a JSON schema validator are provided. This package also supports exporting BioCompute Objects as JSON, PDF, HTML, or 'Word' documents, and exporting to cloud-based platforms.

License AGPL-3

VignetteBuilder knitr


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

Encoding UTF-8

LazyData true

Imports methods, jsonlite, yaml, digest, uuid, jsonvalidate, httr, curl, crayon, cli, stringr, magrittr, rmarkdown

Suggests knitr

RoxygenNote 7.0.0

NeedsCompilation no

Author Nan Xiao [aut, cre] (<https://orcid.org/0000-0002-0250-5673>), Soner Koc [aut], Dennis Dean [aut] (<https://orcid.org/0000-0002-7621-9717>), Seven Bridges Genomics [cph, fnd]

Repository CRAN

Date/Publication 2019-11-28 17:00:02 UTC
R topics documented:

compose_description_v1.3.0 ........................................ 2
compose_error_v1.3.0 ........................................ 5
compose_execution_v1.3.0 .................................... 6
compose_extension_v1.3.0 .................................... 8
compose_fhir_v1.3.0 ........................................ 9
compose_io_v1.3.0 ........................................ 10
compose_parametric_v1.3.0 .................................... 11
compose_provenance_v1.3.0 .................................. 12
compose_scm_v1.3.0 ........................................ 14
compose_tlf_v1.3.0 ........................................ 15
compose_usability_v1.3.0 .................................. 16
compose_v1.3.0 ........................................ 17
convert_json .................................................. 18
convert_yaml .................................................. 19
export_html .................................................. 20
export_json .................................................. 20
export_pdf .................................................. 21
export_sevenbridges ........................................ 22
export_word .................................................. 23
generate_example ........................................... 24
generate_id .................................................. 24
is_bco ......................................................... 25
is_domain ..................................................... 25
validate_checksum_v1.3.0 ................................ 26
validate_schema_v1.3.0 .................................... 26
versions ....................................................... 27

Index 28

compose_description_v1.3.0

Compose BioCompute Object - Description Domain (v1.3.0)

Description

Compose BioCompute Object - Description Domain (v1.3.0)

Usage

compose_description_v1.3.0(
    keywords = NULL,
    xref = NULL,
    platform = "Seven Bridges Platform",
    pipeline_meta = NULL,
    pipeline_prerequisite = NULL,
    pipeline_input = NULL,
compose_description_v1.3.0

pipeline_output = NULL
)

compose_description(
    keywords = NULL,
    xref = NULL,
    platform = "Seven Bridges Platform",
    pipeline_meta = NULL,
    pipeline_prerequisite = NULL,
    pipeline_input = NULL,
    pipeline_output = NULL
)

Arguments

keywords Character vector. A list of keywords to aid in searchability and description of the experiment.

xref Data frame. A list of the databases and/or ontology IDs that are cross-referenced in the BCO.

platform Character string. Reference to a particular deployment of an existing platform where this BCO can be reproduced.

pipeline_meta Data frame. Pipeline metadata. Variables include step_number, name, description, and version.

pipeline_prerequisite Data frame. Packages or prerequisites for running the tools used. Variables include step_number, name, uri, and access_time.

pipeline_input Data frame. Input files for the tools. Variables include step_number, uri, and access_time.

pipeline_output Data frame. Output files for the tools. Variables include step_number, uri, and access_time.

Value

A list of class bco.domain

Examples

keywords <- c("HCV1a", "Ledipasvir", "antiviral resistance", "SNP", "amino acid substitutions")
xref <- data.frame(
    "namespace" = c("pubchem.compound", "pubmed", "so", "taxonomy"),
    "ids" = I(list(67505836, 26508693, c("SO:000002", "SO:0000694", "SO:0000667", "SO:0000045"), 31646)),
    "access_time" = c(
```

platform <- "Seven Bridges Platform"

pipeline_meta <- data.frame(
  "step_number" = c("1"),
  "name" = c("HIVE-hexagon"),
  "description" = c("Alignment of reads to a set of references"),
  "version" = c("1.3"),
  stringsAsFactors = FALSE
)

pipeline_prerequisite <- data.frame(
  "step_number" = rep("1", 5),
  "name" = c("Hepatitis C virus genotype 1",
             "Hepatitis C virus type 1b complete genome",
             "Hepatitis C virus (isolate JFH-1) genomic RNA",
             "Hepatitis C virus clone J8CF, complete genome",
             "Hepatitis C virus S52 polyprotein gene"),
  "access_time" = c(as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"),
                    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"),
                    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"),
                    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"),
                    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST")),
  stringsAsFactors = FALSE
)

pipeline_input <- data.frame(
  "step_number" = rep("1", 2),
  "uri" = c("https://example.com/dna.cgi?cmd=objFile&ids=514683",
            "https://example.com/dna.cgi?cmd=objFile&ids=514682"),
  "access_time" = c(as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"),
                    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"))
)
```
compose_error_v1.3.0

compose_description(  
  keywords, xref, platform,  
  pipeline_meta, pipeline_prerequisite, pipeline_input, pipeline_output  
) %>% convert_json()

---

**Description**

The error domain can be used to determine what range of input returns outputs that are within the tolerance level defined in this subdomain and therefore can be used to optimize algorithm (domain definition).

**Usage**

compose_error_v1.3.0(empirical = NULL, algorithmic = NULL)

compose_error(empirical = NULL, algorithmic = NULL)

**Arguments**

- **empirical**
  - Data frame. Variables include key and value. Each row is one item in the empirical error subdomain.

- **algorithmic**
  - Data frame. Variables include key and value. Each row is one item in the algorithmic subdomain.

**Value**

A list of class bco.domain
Examples

```r
empirical <- data.frame(
  "key" = c("false_negative_alignment_hits", "false_discovery"),
  "value" = c("<0.0010", "<0.05"),
  stringsAsFactors = FALSE
)

algorithmic <- data.frame(
  "key" = c("false_positive_mutation_calls", "false_discovery"),
  "value" = c("<0.00005", "0.005"),
  stringsAsFactors = FALSE
)

compose_error(empirical, algorithmic) %>% convert_json()
```

---

**Description**

Compose BioCompute Object - Execution Domain (v1.3.0)

**Usage**

```r
compose_execution_v1.3.0(
  script = NULL,
  script_driver = NULL,
  software_prerequisites = NULL,
  external_data_endpoints = NULL,
  environment_variables = NULL
)
```

```r
compose_execution(
  script = NULL,
  script_driver = NULL,
  software_prerequisites = NULL,
  external_data_endpoints = NULL,
  environment_variables = NULL
)
```

**Arguments**

- `script` Character string. Points to internal or external references to an object that was used to perform computations for this BCO instance.
- `script_driver` Character string. Indicate what kind of executable can be launched in order to perform a sequence of commands described in the script in order to run the pipeline.
compose_execution_v1.3.0

software_prerequisites
Data frame. The minimal necessary prerequisites, library, and tool versions needed to successfully run the script to produce BCO. Variables include name, version, uri, access_time, and sha1_chksum. Each row is one item in the output subdomain.

external_data_endpoints
Data frame. The minimal necessary domain-specific external data source access to successfully run the script to produce the BCO. Variables include mediatype, name, and url. Each row is one item in the output subdomain.

environment_variables
Data frame. Key-value pairs useful to configure the execution environment on the target platform. Variables include key and value.

Value
A list of class bco.domain

Examples

script <- "https://example.com/workflows/antiviral_resistance_detection_hive.py"
script_driver <- "shell"
software_prerequisites <- data.frame(
  "name" = c("HIVE-hexagon", "HIVE-heptagon"),
  "version" = c("babajanian.1", "albinoni.2"),
  "uri" = c(
    "https://example.com/dna.cgi?cmd=dna-hexagon&cmdMode=-",
    "https://example.com/dna.cgi?cmd=dna-heptagon&cmdMode=-"
  ),
  "access_time" = c(
    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%dT%H:%M:%S", tz = "EST"),
    as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%dT%H:%M:%S", tz = "EST")
  ),
  "sha1_chksum" = c("d60f506cddac09e9e816531e7905ca1ca664f3c", NA),
  stringsAsFactors = FALSE)

external_data_endpoints <- data.frame(
  "name" = c("generic name", "access to ftp server", "access to e-utils web service"),
  "url" = c(
    "protocol://domain:port/application/path",
    "ftp://data.example.com:21/",
  ),
  stringsAsFactors = FALSE)

environment_variables <- data.frame(
  "key" = c("HOSTTYPE", "EDITOR"),
  "value" = c("x86_64-linux", "vim")
)

compose_execution(
  script, script_driver, software_prerequisites, external_data_endpoints, environment_variables
) %>% convert_json()
Compose BioCompute Object - Extension Domain (v1.3.0)

Description
Compose BioCompute Object - Extension Domain (v1.3.0)

Usage
compose_extension_v1.3.0(fhir = NULL, scm = NULL)

compose_extension(fhir = NULL, scm = NULL)

Arguments
fhir
FHIR extension domain composed by compose_fhir.

scm
SCM extension domain composed by compose_scm.

Value
A list of class bco.domain

Examples
fhir_endpoint <- "https://fhirtest.uhn.ca/baseDstu3"
fhir_version <- "3"
fhir_resources <- data.frame(
  "id" = c("21376", "6288583", "25544", "92440", "4588936"),
  "resource" = c(
    "Sequence", "DiagnosticReport", "ProcedureRequest",
    "Observation", "FamilyMemberHistory"
  ),
  stringsAsFactors = FALSE
)
fhir <- compose_fhir(fhir_endpoint, fhir_version, fhir_resources)

scm_repository <- "https://github.com/example/repo"
scm_type <- "git"
scm_commit <- "c9ffe0b60fa3bce8e138af7c99ca141a6bb8fb21"
scm_path <- "workflow/hive-viral-mutation-detection.cwl"
scm_preview <- "https://github.com/example/repo/blob/master/mutation-detection.cwl"

scm <- compose_scm(scm_repository, scm_type, scm_commit, scm_path, scm_preview)

compose_extension(fhir, scm) %>% convert_json()
Description

Compose BioCompute Object - FHIR Extension (v1.3.0)

Usage

compose_fhir_v1.3.0(endpoint = NULL, version = NULL, resources = NULL)

compose_fhir(endpoint = NULL, version = NULL, resources = NULL)

Arguments

endpoint Character string. The URL of the endpoint of the FHIR server containing the resource.

version Character string. The FHIR version used.

resources Data frame with two variables: id and resource. Each row is one item of resources to fetch from the endpoint.

Value

A list of class bco.domain

Examples

fhir_endpoint <- "https://fhirtest.uhn.ca/baseDstu3"
fhir_version <- "3"
fhir_resources <- data.frame(
  "id" = c("21376", "6288583", "25544", "92440", "4588936"),
  "resource" = c("Sequence", "DiagnosticReport", "ProcedureRequest", 
                 "Observation", "FamilyMemberHistory" ),
  stringsAsFactors = FALSE
)

compose_fhir(fhir_endpoint, fhir_version, fhir_resources) %>% convert_json()
compose_io_v1.3.0

Compose BioCompute Object - Input and Output Domain (v1.3.0)

Description
This domain contains the list of global input and output files created by the computational workflow, excluding the intermediate files.

Usage
compose_io_v1.3.0(input = NULL, output = NULL)

compose_io(input = NULL, output = NULL)

Arguments

input
Data frame. Variables include filename, uri, and access_time. Each row is one item in the input subdomain.

output
Data frame. Variables include mediatype, uri, and access_time. Each row is one item in the output subdomain.

Value
A list of class bco.domain

Examples

input_subdomain <- data.frame(
  "filename" = c("Hepatitis C virus genotype 1", "Hepatitis C virus type 1b complete genome"),
  "access_time" = c(as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d%H:%M:%S", tz = "EST"), as.POSIXct("2017-01-24T09:40:17", format = "%Y-%m-%d%H:%M:%S", tz = "EST"))
),
stringsAsFactors = FALSE
)

output_subdomain <- data.frame(
  "mediatype" = c("text/csv", "text/csv"),
  "uri" = c("https://example.com/data/514769/dnaAccessionBased.csv", "https://example.com/data/514801/SNPProfile*.csv"),
)
compose_parametric_v1.3.0

Compose BioCompute Object - Parametric Domain (v1.3.0)

Description
Non-default parameters customizing the computational flow which can affect the output of the calculations (domain definition).

Usage
compose_parametric_v1.3.0(df = NULL)
compose_parametric(df = NULL)

Arguments

df
Data frame. Variables include param (parameter names), value (value of the parameters), and step (step number for each parameter).

Value
A list of class bco.domain

Examples

df_parametric <- data.frame(
  "param" = c(
    "seed", "minimum_match_len",
    "divergence_threshold_percent",
    "minimum_coverage", "freq_cutoff"
  ),
  "value" = c("14", "66", "0.30", "15", "0.10"),
  "step" = c(1, 1, 1, 2, 2)
)

compose_parametric(df_parametric) %>% convert_json()
Description
Compose BioCompute Object - Provenance Domain (v1.3.0)

Usage
compose_provenance_v1.3.0(
    name = NULL,
    version = NULL,
    review = NULL,
    derived_from = NULL,
    obsolete_after = NULL,
    embargo = NULL,
    created = NULL,
    modified = NULL,
    contributors = NULL,
    license = NULL
)

compose_provenance(
    name = NULL,
    version = NULL,
    review = NULL,
    derived_from = NULL,
    obsolete_after = NULL,
    embargo = NULL,
    created = NULL,
    modified = NULL,
    contributors = NULL,
    license = NULL
)

Arguments
name Character string. Name for the BCO.
version Character string. Version of this BCO instance object. Should follow the Semantic Versioning format (MAJOR.MINOR.PATCH).
review Data frame. Reviewer identifiers and descriptions of the status of an object in the review process.
derived_from Character string. Inheritance/derivation description.
obsolete_after Date-time object. Expiration date of the object (optional).
embargo Vector of date-time objects `start_time` and `end_time`. If the object has a period of time that it is not public, that range can be specified with this.

created Date-time object. Initial creation time of the object.

modified Date-time object. The most recent modification time of the object.

contributors Data frame. Contributor identifiers and descriptions of their contribution types.

license Character string. Licence URL or other licence information (text).

Value
A list of class `bco.domain`

Examples

```r
name <- "HCV1a ledipasvir resistance SNP detection"
version <- "1.0.0"
review <- data.frame(
  "status" = c("approved", "approved"),
  "reviewer_comment" = c("Approved by [company name] staff. Waiting for approval from FDA Reviewer", "The revised BCO looks fine"),
  "date" = c(as.POSIXct("2017-11-12T12:30:48", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"), as.POSIXct("2017-12-12T12:30:48", format = "%Y-%m-%d\T%H:%M:%S", tz = "America/Los_Angeles")),
  "reviewer_name" = c("Jane Doe", "John Doe"),
  "reviewer_affiliation" = c("Seven Bridges Genomics", "U.S. Food and Drug Administration"),
  "reviewer_email" = c("example@sevenbridges.com", "example@fda.gov"),
  "reviewer_contribution" = c("curatedBy", "curatedBy"),
  "reviewer_orcid" = c("https://orcid.org/0000-0000-0000-0000", NA),
  stringsAsFactors = FALSE)

derived_from <- "https://github.com/biocompute-objects/BCO_Specification/blob/1.2.1-beta/HCV1a.json"
obsolete_after <- as.POSIXct("2018-11-12T12:30:48", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST")

embargo <- c(
  "start_time" = as.POSIXct("2017-10-12T12:30:48", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"),
  "end_time" = as.POSIXct("2017-11-12T12:30:48", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST"))

created <- as.POSIXct("2017-01-20T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST")
modified <- as.POSIXct("2019-05-10T09:40:17", format = "%Y-%m-%d\T%H:%M:%S", tz = "EST")

contributors <- data.frame(
  "name" = c("Jane Doe", "John Doe"),
  "affiliation" = c("Seven Bridges Genomics", "U.S. Food and Drug Administration"),
  "email" = c("example@sevenbridges.com", "example@fda.gov"),
  "contribution" = I(list(c("createdBy", "curatedBy"), c("authoredBy")))),
  "orcid" = c("https://orcid.org/0000-0000-0000-0000", NA),
compose_scm_v1.3.0

Compose BioCompute Object - SCM Extension (v1.3.0)

Description

Compose BioCompute Object - SCM Extension (v1.3.0)

Usage

compose_scm_v1.3.0(
    scm_repository = NULL,
    scm_type = c("git", "svn", "hg", "other"),
    scm_commit = NULL,
    scm_path = NULL,
    scm_preview = NULL
)    

compose_scm(
    scm_repository = NULL,
    scm_type = c("git", "svn", "hg", "other"),
    scm_commit = NULL,
    scm_path = NULL,
    scm_preview = NULL
    
)

Arguments

scm_repository Character string. Base URL of the SCM repository.

scm_type Character string. Type of SCM database. Must be one of "git", "svn", "hg", or "other".

scm_commit Character string. Revision within the SCM repository. Should be a repository-wide commit identifier or name of a tag, but may be a name of a branch.

scm_path Character string. Path from the repository to the source code referenced. Should not start with /.

scm_preview Character string. The full URI for the source code referenced by the BioCompute Object.
Value

A list of class bco.domain

Examples

```r
cscm_repository <- "https://github.com/example/repo"
cscm_type <- "git"
cscm_commit <- "c9ffe06060fa3b8e138af7c99ca141a6b8fb21"
cscm_path <- "workflow/hive-viral-mutation-detection.cwl"
cscm_preview <- "https://github.com/example/repo/blob/master/mutation-detection.cwl"

compose_scm(scm_repository, scm_type, scm_commit, scm_path, scm_preview) %>% convert_json()
```

Description

Compose BioCompute Object - Top Level Fields (v1.3.0)

Usage

```r
compose_tlf_v1.3.0(
  provenance,
  usability,
  extension,
  description,
  execution,
  parametric,
  io,
  error,
  bco_id = NULL
)
```

```r
compose_tlf(
  provenance,
  usability,
  extension,
  description,
  execution,
  parametric,
  io,
  error,
  bco_id = NULL
)
```
**compose_usability_v1.3.0**

**Arguments**

- **provenance**  Provenance domain
- **usability**  Usability domain
- **extension**  Extension domain
- **description**  Description domain
- **execution**  Execution domain
- **parametric**  Parametric domain
- **io**  I/O domain
- **error**  Error domain
- **bco_id**  BioCompute Object identifier (definition). If NULL, will use a UUID generated by `generate_id`.

**Value**

A vector of top level fields

**Examples**

```r
compose_tlf(
  compose_provenance(), compose_usability(), compose_extension(),
  compose_description(), compose_execution(), compose_parametric(),
  compose_io(), compose_error()
) %>% convert_json()
```

---

**compose_usability_v1.3.0**

*Compose BioCompute Object - Usability Domain (v1.3.0)*

**Description**

The usability domain (domain definition).

**Usage**

```r
compose_usability_v1.3.0(text = NULL)
compose_usability(text = NULL)
```

**Arguments**

- **text**  A character vector of free text values that could improves search-ability, provide specific scientific use cases, and a description of the function of the object.

**Value**

A list of class `bco.domain`
Examples

```r
text <- c(
  paste("Identify baseline single nucleotide polymorphisms (SNPs)[SO:0000694]", 
        "(insertions)[SO:0000667], and (deletions)[SO:0000045] that correlate", 
        "with reduced (ledipasvir)[pubchem.compound:67505836] antiviral drug", 
        "efficacy in (Hepatitis C virus subtype 1)[taxonomy:31646]"
  ),
  paste("Identify treatment emergent amino acid (substitutions)[SO:1000002]", 
        "that correlate with antiviral drug treatment failure"
  ),
  paste("Determine whether the treatment emergent amino acid", 
        "(substitutions)[SO:1000002] identified correlate with treatment", 
        "failure involving other drugs against the same virus"
  )
)

text %>%
  compose_usability() %>%
  convert_json()
```

Description

Compose BioCompute Object (v1.3.0)

Usage

```r
compose_v1.3.0(
  tlf, 
  provenance, 
  usability, 
  extension, 
  description, 
  execution, 
  parametric, 
  io, 
  error 
)
```

```r
compose(
  tlf, 
  provenance, 
  usability,
```
extension, description, execution, parametric, io, error
)

Arguments

tlf          Top level fields
provenance   Provenance domain
usability     Usability domain
extension     Extension domain
description   Description domain
execution     Execution domain
parametric    Parametric domain
io            I/O domain
error         Error domain

Value

A list of class bco

Examples

tlf <- compose_tlf(
    compose_provenance(), compose_usability(), compose_extension(),
    compose_description(), compose_execution(), compose_parametric(),
    compose_io(), compose_error()
)
biocompute::compose(
    tlf,
    compose_provenance(), compose_usability(), compose_extension(),
    compose_description(), compose_execution(), compose_parametric(),
    compose_io(), compose_error()
) %>% convert_json()

convert_json

Convert BioCompute Object or domain to JSON string

Description

Convert BioCompute Object or domain to JSON string
convert_yaml

Usage

```r
convert_json(x, pretty = TRUE, auto_unbox = TRUE, na = "string", ...)
```

Arguments

- `x`: BioCompute Object or domain
- `pretty`: Prettify the JSON string? Default is `TRUE`.
- `auto_unbox`: Unbox all atomic vectors of length 1? Default is `TRUE`.
- `na`: How to represent NA values: must be "null" or "string". Default is "string".
- `...`: Additional parameters for `toJSON`.

Value

JSON string of the BioCompute Object

Examples

```r
compose_description() %>% convert_json()
generate_example("minimal") %>% convert_json()
```
export_html  
*Export BioCompute Object as HTML*

**Description**

Export BioCompute Object as HTML

**Usage**

```r
export_html(x, file, wrap = FALSE, linewidth = 80, ...)
```

**Arguments**

- **x**: BioCompute Object JSON string from `convert_json`
- **file**: HTML output file path
- **wrap**: Should the long lines be wrapped?
- **linewidth**: Maximum linewidth when `wrap` is `TRUE`
- **...**: Additional parameters for `render`

**Value**

Path to the output file

**Examples**

```r
file_html <- tempfile(fileext = ".html")
generate_example("HCV1a") %>%
  convert_json() %>%
  export_html(file_html)
```

export_json  
*Export BioCompute Object as JSON*

**Description**

Export BioCompute Object as JSON

**Usage**

```r
export_json(x, file)
```
export_pdf

Arguments

x  BioCompute Object JSON string from convert_json

file  JSON file path

Value

Path to the output file

Examples

```r
file_json <- tempfile(fileext = ".json")
generate_example("HCV1a") %>%
  convert_json() %>%
  export_json(file_json)
cat(paste(readLines(file_json), collapse = "\n"))
```

---

export_pdf  Export BioCompute Object as PDF

Description

Export BioCompute Object as PDF

Usage

```r
export_pdf(x, file, wrap = FALSE, linewidth = 80, ...)
```

Arguments

x  BioCompute Object JSON string from convert_json

file  PDF output file path

wrap  Should the long lines be wrapped?

linewidth  Maximum linewidth when wrap is TRUE.

...  Additional parameters for render.

Value

Path to the output file

Examples

```r
file_pdf <- tempfile(fileext = ".pdf")
generate_example("HCV1a") %>%
  convert_json() %>%
  export_pdf(file_pdf)
```
export_sevenbridges  Export BioCompute Object to Seven Bridges Platforms

Description

Export BioCompute Object to Seven Bridges Platforms

Usage

```r
export_sevenbridges(
    file, 
    name = NULL, 
    project = NULL, 
    token = NULL, 
    base_url = "https://api.sbgenomics.com/v2/", 
    overwrite = TRUE
)
```

Arguments

- **file**: Path to the BCO file.
- **name**: Name of the BCO file to create on the platform. Defaults to the name of the input file.
- **project**: Project to upload (export) the BCO file to. Format: "username/project".
- **token**: API auth token for the platform. Generate the token from the platform’s Developer Dashboard.
- **base_url**: API base URL. Get the base URL from the platform’s Developer Dashboard.
- **overwrite**: If TRUE, will overwrite the existing BCO file with the same name in that project (if any). If FALSE, will not overwrite.

Value

Response of the file upload request

Examples

```r
file_json <- tempfile(fileext = ".json")
generate_example("HCV1a") %>%
    convert_json() %>%
    export_json(file_json)

export_sevenbridges(
    file_json,
    project = "rosalind_franklin/project_name",
    token = "your_api_auth_token",
    base_url = "https://cgc-api.sbgenomics.com/v2/"
)
**export_word**  

Export BioCompute Object as Word document

**Description**

Export BioCompute Object as Word document

**Usage**

```r
export_word(x, file, wrap = FALSE, linewidth = 80, ...) 
```

**Arguments**

- `x`: BioCompute Object JSON string from `convert_json`
- `file`: Word (docx) output file path
- `wrap`: Should the long lines be wrapped?
- `linewidth`: Maximum linewidth when `wrap` is `TRUE`.
- `...`: Additional parameters for `render`.

**Value**

Path to the output file

**Examples**

```r
file_docx <- tempfile(fileext = "\.docx") 
generate_example("HCV1a") %>% 
  convert_json() %>% 
  export_word(file_docx)
```
**generate_example**  
*Generate example BioCompute Objects*

**Description**
Generate example BioCompute Objects

**Usage**
generate_example(type = c("minimal", "HCV1a"))

**Arguments**
- **type**  
  Example type. Default is "minimal".

**Value**
Example BioCompute Object

**Examples**
generate_example("minimal") %>% convert_json()

---

**generate_id**  
*Generate ID for the BioCompute Object*

**Description**
Generate ID for the BioCompute Object

**Usage**
generate_id(platform = c("sevenbridges"))

**Arguments**
- **platform**  
  Platform. Default is "sevenbridges".

**Value**
BioCompute Object ID

**Examples**
generate_id()
**is_bco**

Is this a BCO object?

**Description**

Is this a BCO object?

**Usage**

```r
is_bco(x)
```

**Arguments**

- `x` any object

**Value**

Logical. TRUE if it is a BCO object, FALSE if not.

**Examples**

```r
generate_example("minimal") %>% is_bco()
```

---

**is_domain**

Is this a domain object?

**Description**

Is this a domain object?

**Usage**

```r
is_domain(x)
```

**Arguments**

- `x` any object

**Value**

Logical. TRUE if it is a domain object, FALSE if not.

**Examples**

```r
is_domain(compose_description())
```
validate_checksum_v1.3.0

BioCompute Objects checksum validator (v1.3.0)

Description

BioCompute Objects checksum validator (v1.3.0)

Usage

validate_checksum_v1.3.0(file)

validate_checksum(file)

Arguments

file Path to the BCO JSON file

Value

Logical. TRUE if the checksum matched, FALSE if not.

Note

An SHA-256 checksum is calculated and stored in the top level fields when a BioCompute Object is created. In reality, due to the delicate differences in how the data in JSON is represented, parsed, and handled in different languages, there could be false positives in the validation results.

Examples

bco <- tempfile(fileext = ".json")
generate_example("HCV1a") %>%
  convert_json() %>%
  export_json(bco)
bco %>% validate_checksum()

validate_schema_v1.3.0

BioCompute Objects schema validator (v1.3.0)

Description

BioCompute Objects schema validator (v1.3.0)
Usage

validate_schema_v1.3.0(file)

validate_schema(file)

Arguments

file Path to the BCO JSON file

Value

None

Note

JSON schema validators for BCO domains and complete BCO based on jsonvalidate. Refer to the BioCompute Objects Schema for specific JSON schemas.

Examples

bco <- tempfile(fileext = "json")
generate_example("HCV1a") %>%
  convert_json() %>%
  export_json(bco)
bco %>% validate_schema()

versions

BioCompute Object specification versions

Description

BioCompute Object specification versions

Usage

versions()

Value

List of current and all available BioCompute Object specification versions supported by the package.

Examples

versions()
Index

as.yaml, 19

compose (compose_v1.3.0), 17
compose_description
  (compose_description_v1.3.0), 2
compose_description_v1.3.0, 2
compose_error (compose_error_v1.3.0), 5
compose_error_v1.3.0, 5
compose_execution
  (compose_execution_v1.3.0), 6
compose_execution_v1.3.0, 6
compose_extension
  (compose_extension_v1.3.0), 8
compose_extension_v1.3.0, 8
compose_fhir (compose_fhir_v1.3.0), 9
compose_fhir_v1.3.0, 9
compose_io (compose_io_v1.3.0), 10
compose_io_v1.3.0, 10
compose_parametric
  (compose_parametric_v1.3.0), 11
compose_parametric_v1.3.0, 11
compose_provenance
  (compose_provenance_v1.3.0), 12
compose_provenance_v1.3.0, 12
compose_scm, 8
compose_scm (compose_scm_v1.3.0), 14
compose_scm_v1.3.0, 14
compose_tlf (compose_tlf_v1.3.0), 15
compose_tlf_v1.3.0, 15
compose_usability
  (compose_usability_v1.3.0), 16
compose_usability_v1.3.0, 16
compose_v1.3.0, 17
convert_json, 18, 20, 21, 23
convert_yaml, 19

eexport_sevenbridges, 22
export_word, 23

generate_example, 24
generate_id, 16, 24

is_bco, 25
is_domain, 25

render, 20, 21, 23
toJSON, 19

validate_checksum
  (validate_checksum_v1.3.0), 26
validate_checksum_v1.3.0, 26
validate_schema
  (validate_schema_v1.3.0), 26
validate_schema_v1.3.0, 26
versions, 27

export_html, 20
export_json, 20
export_pdf, 21