Package 'rDataPipeline'

November 17, 2021

Title Functions to Interact with the 'FAIR Data Pipeline'

Version 0.54.1

Description R implementation of the 'FAIR Data Pipeline API'. The 'FAIR Data Pipeline' is intended to enable tracking of provenance of FAIR (findable, accessible and interoperable) data used in epidemiological modelling.

License GPL (>= 3)

Imports assertthat, cli, configr, dplyr, git2r, httr, jsonlite, openssl, R6, rhdf5, semver, stats, usethis, utils, yaml

Suggests units, testthat

biocViews rhdf5

Encoding UTF-8

RoxygenNote 7.1.2

URL https://www.fairdatapipeline.org/rDataPipeline/, https://github.com/FAIRDataPipeline/rDataPipeline

BugReports https://github.com/FAIRDataPipeline/rDataPipeline/issues

NeedsCompilation no

Author Sonia Mitchell [cre, aut] (<https://orcid.org/0000-0003-1536-2066>), Ryan Field [ctb] (<https://orcid.org/0000-0002-4424-9890>)

Maintainer Sonia Mitchell <sonia.mitchell@glasgow.ac.uk>

Repository CRAN

Date/Publication 2021-11-17 21:00:06 UTC

R topics documented:

`rDataPipeline-package` *rDataPipeline*

Description

FAIR Data Pipeline API

Details

For more information see https://www.fairdatapipeline.org/

add_read *add_read*

Description

Add data product to read block of user-written config file. Used in combination with create_config() for unit testing.

Usage

```
add_read(
  path,
  data_product,
  component,
  version,
  use_data_product,
  use_component,
  use_version,
  use_namespace
)
```

Arguments

path	config file path
data_product	data_product field
component	component field
version	(optional) version field
use_data_product	(optional) use_data_product field
use_component	(optional) use_component field
use_version	(optional) use_version field
use_namespace	(optional) use_namespace field

Examples

```
## Not run:
path <- "test_config/config.yaml"

# Write run_metadata block
create_config(path = path,
              description = "test",
              input_namespace = "test_user",
              output_namespace = "test_user")

# Write read block
add_read(path = path,
```

```
          data_product = "test/array",
          component = "level/a/s/d/f/s",
          version = "0.2.0")

## End(Not run)
```

`add_write` *add_write*

Description

Add data product to `read` block of user-written config file. Used in combination with `create_config()` for unit testing.

Usage

```
add_write(
  path,
  data_product,
  description,
  version,
  file_type,
  use_data_product,
  use_component,
  use_version,
  use_namespace
)
```

Arguments

`path`	config file path
`data_product`	data_product field
`description`	component field
`version`	(optional) version field
`file_type`	(optional) file type field
`use_data_product`	(optional) use_data_product field
`use_component`	(optional) use_component field
`use_version`	(optional) use_version field
`use_namespace`	(optional) use_namespace field

Examples

```
## Not run:
path <- "test_config/config.yaml"

# Write run_metadata block
create_config(path = path,
              description = "test",
              input_namespace = "test_user",
              output_namespace = "test_user")

# Write read block
add_write(path = path,
          data_product = "test/array",
          description = "data product description",
          version = "0.2.0")

## End(Not run)
```

`check_config` *check_config*

Description

check_config

Usage

```
check_config(handle, data_product, what)
```

Arguments

`handle`	an object of class `fdp,R6` containing metadata required by the Data Pipeline API
`data_product`	data_product
`what`	what

`check_dataproduct_exists` *check_dataproduct_exists*

Description

If a data product already exists with the same name, version, and namespace, throw an error

Usage

```
check_dataproduct_exists(
  write_dataproduct,
  write_version,
  write_namespace_id,
  endpoint
)
```

Arguments

write_dataproduct	write_dataproduct
write_version	write_version
write_namespace_id	write_namespace_id
endpoint	endpoint

`check_datetime` *check_datetime*

Description

check_datetime

Usage

```
check_datetime(table, this_field, query_class, this_query)
```

Arguments

table	a string specifying the name of the table
this_field	a string specifying the name of the field
query_class	a string specifying the class of the field
this_query	a string specifying the contents of the field

check_exists *Check if entry exists in the data registry*

Description

Check whether an entry already exists in a table (in the data registry)

Usage

```
check_exists(table, query)
```

Arguments

table	a string specifying the name of the table
query	a list containing a valid query for the table, *e.g.* list(field = value)

Value

Returns TRUE if the entry exists and FALSE if it doesn't

check_field *check_field*

Description

check_field

Usage

```
check_field(table, this_field, query_class, this_query, method, endpoint)
```

Arguments

table	a string specifying the name of the table
this_field	a string specifying the name of the field
query_class	a string specifying the class of the field
this_query	a string specifying the contents of the field
method	a string specifying the method, c("GET", "POST")
endpoint	endpoint

check_fields *check_fields*

Description

check_fields

Usage

```
check_fields(table, query, method, endpoint)
```

Arguments

table	a string specifying the name of the table
query	a list containing the query
method	a string specifying the method, c("GET", "POST")
endpoint	endpoint

check_handle *check_handle*

Description

check_handle

Usage

```
check_handle(handle, data_product, what, component)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the name of the data product
what	element in handle – one of c("inputs", "outputs")
component	a string specifying the name of the component

check_integer *check_integer*

Description

check_integer

Usage

```
check_integer(table, this_field, query_class, this_query)
```

Arguments

table	a string specifying the name of the table
this_field	a string specifying the name of the field
query_class	a string specifying the class of the field
this_query	a string specifying the contents of the field

check_local_repo *check_local_repo*

Description

check_local_repo

Usage

```
check_local_repo(path)
```

Arguments

path	Local repository file path

Value

boolean, if local repository is clean (TRUE, else FALSE)

check_string *check_string*

Description

check_string

Usage

```
check_string(table, this_field, this_query)
```

Arguments

table	a string specifying the name of the table
this_field	a string specifying the name of the field
this_query	a string specifying the contents of the field

check_table_exists *Check if table exists*

Description

Check if table exists in the data registry

Usage

```
check_table_exists(table)
```

Arguments

table	a string specifying the name of the table

Value

Returns TRUE if a table exists, FALSE if it doesn't

check_yaml_write *check_yaml_write*

Description

check_yaml_write

Usage

```
check_yaml_write(handle, data_product)
```

Arguments

handle	fdp object
data_product	a string specifying the name of the data product

clean_query *Clean query*

Description

Function to clean a query and return it without an api prefix

Usage

```
clean_query(data, endpoint)
```

Arguments

data	a list containing a valid query for the table, *e.g.* list(field = value)
endpoint	endpoint

create_config *create_config*

Description

Generates (user generated) config.yaml files for unit tests. Use `add_read()` and `add_write()` functions to add read and write blocks.

Usage

```
create_config(
  path,
  description,
  input_namespace,
  output_namespace,
  write_data_store = file.path(tempdir(), "datastore", ""),
  force = TRUE,
  local_repo = "local_repo"
)
```

Arguments

`path`	config file path
`description`	description field
`input_namespace`	input_namespace field
`output_namespace`	output_namespace field
`write_data_store`	write_data_store field
`force`	force
`local_repo`	local_repo

create_index *create_index*

Description

create_index

Usage

```
create_index(self)
```

Arguments

`self`	self

`create_version_number` *Create version number*

Description

Creates a version number from either *a date and a version* **or** *a date and major and patch* **or** *major minor patch*. If no parameters are supplied a default version is returned `0.1.0` This function prioritizes download date and version over all other parameters

Usage

```
create_version_number(
  download_date = NULL,
  version = NULL,
  major = 0,
  minor = "1",
  patch = 0
)
```

Arguments

`download_date`	(Optional) download_date This can either be a date or datetime but must include the full year *e.g* from `Sys.Date()` `2020-01-01` Or from `Sys.time()` `2020-01-01 00:00:00 BST` note: also accepts / delimited dates *e.g* `01/02/2020` or `2020/01/01` and accepts date without delimiters but assumes ddmmyyyy or yyyymmdd *e.g.* `20200201`
`version`	version number using major.minor.patch numbering *e.g.* `0.1.0` or major.patch *e.g.* `0.0`
`major`	major number if not using `version`
`minor`	minor number if not using `date`
`patch`	patch number if not using `version`

Value

returns a character vector in the format of `major.minor.patch` *e.g.* `0.20200101.0`

`download_from_database`

Download source file from database

Description

Download source file from database

Usage

```
download_from_database(
  source_root,
  source_path,
  path,
  filename,
  overwrite = FALSE
)
```

Arguments

source_root	a string specifying the source root
source_path	a string specifying the source path
path	a string specifying where the file will be saved
filename	a string specifying the filename (the name given to the saved file)
overwrite	a boolean specifying whether or not the file should be overwritten if it already exists

See Also

Other download functions: download_from_url()

---

download_from_url *Download source file from URL*

Description

This function will download a file from a url

Usage

```
download_from_url(source_root, source_path, path, filename)
```

Arguments

source_root	a string specifying the source root
source_path	a string specifying the source path
path	a string specifying where the file will be saved
filename	a string specifying the filename (the name given to the saved file)

See Also

Other download functions: download_from_database()

extract_id *extract_id*

Description

extract_id

Usage

```
extract_id(url, endpoint)
```

Arguments

url	url
endpoint	endpoint

fair_init *fair_init*

Description

fair_init

Usage

```
fair_init(name, identifier, endpoint = "http://localhost:8000/api/")
```

Arguments

name	a string specifying the full name or organisation name of the author; note that at least one of name or identifier must be specified
identifier	(optional) a string specifying the full URL identifier (*e.g.* ORCiD or ROR ID) of the author
endpoint	a string specifying the registry endpoint

fair_run *fair_run*

Description

fair_run

Usage

```
fair_run(
  path = "config.yaml",
  endpoint = "http://localhost:8000/api/",
  skip = FALSE
)
```

Arguments

path	string
endpoint	a string specifying the registry endpoint
skip	don't bother checking whether the repo is clean

fdp-class *fdp-class*

Description

fdp-class

fdp-class

Details

Container for class fdp

Public fields

yaml a list containing the contents of the working config.yaml

fdp_config_dir a string specifying the directory passed from fair run

model_config a string specifying the URL of an entry in the object table associated with the storage_location of the working config.yaml

submission_script a string specifying the URL of an entry in the object table associated with the storage_location of the submission script

code_repo a string specifying the URL of an entry in the object table associated with the GitHub repository

code_run a string specifying the URL of an entry in the code_run table

inputs a data.frame containing metadata associated with code_run inputs

outputs a data.frame containing metadata associated with code_run outputs

issues a data.frame containing metadata associated with code_run issues

Methods

Public methods:

- fdp$new()
- fdp$print()
- fdp$input()
- fdp$output()
- fdp$output_index()
- fdp$raise_issue()
- fdp$finalise_output_hash()
- fdp$finalise_output_url()
- fdp$clone()

Method new(): Create a new fdp object

Usage:

```
fdp$new(
  yaml,
  fdp_config_dir,
  model_config,
  submission_script,
  code_repo,
  code_run
)
```

Arguments:

yaml a list containing the contents of the working config.yaml

fdp_config_dir a string specifying the directory passed from fair run

model_config a string specifying the URL of an entry in the object table associated with the storage_location of the working config.yaml

submission_script a string specifying the URL of an entry in the object table associated with the storage_location of the submission script

code_repo a string specifying the URL of an entry in the object table associated with the GitHub repository

code_run a string specifying the URL of an entry in the code_run table

Returns: Returns a new fdp object

Method print(): Print method

Usage:

```
fdp$print(...)
```

Method `input()`**:** Record `code_run` inputs in `fdp` object

Usage:

```
fdp$input(
  data_product,
  use_data_product,
  use_component,
  use_version,
  use_namespace,
  path,
  component_url
)
```

Arguments:

`data_product` a `string` specifying the name of the data product, used as a reference

`use_data_product` a `string` specifying the name of the data product, used as input in the `code_run`

`use_component` a `string` specifying the name of the data product component, used as input in the `code_run`

`use_version` a `string` specifying the data product version, used as input in the `code_run`

`use_namespace` a `string` specifying the namespace in which the data product resides, used as input in the `code_run`

`path` a `string` specifying the location of the data product in the local data store

`component_url` a `string` specifying the URL of an entry in the `object_component` table

Returns: Returns an updated `fdp` object

Method `output()`**:** Record `code_run` outputs in `fdp` object

Usage:

```
fdp$output(
  data_product,
  use_data_product,
  use_component,
  use_version,
  use_namespace,
  path,
  data_product_description,
  component_description,
  public
)
```

Arguments:

`data_product` a `string` specifying the name of the data product, used as a reference

`use_data_product` a `string` specifying the name of the data product, used as output in the `code_run`

`use_component` a `string` specifying the name of the data product component, used as output in the `code_run`

`use_version` a `string` specifying the version of the data product, used as output in the `code_run`

use_namespace a string specifying the namespace in which the data product resides, used as output in the code_run

path a string specifying the location of the data product in the local data store

data_product_description a string containing a description of the data product

component_description a string containing a description of the data product component

public

Returns: Returns an updated fdp object

Method output_index(): Return index of data product recorded in fdp object so that an issue may be attached

Usage:

```
fdp$output_index(data_product, component, version, namespace)
```

Arguments:

data_product a string specifying the name of the data product, used as output in the code_run

component a string specifying the name of the data product component, used as output in the code_run

version a string specifying the name of the data product version, used as output in the code_run

namespace a string specifying the namespace in which the data product resides, used as input in the code_run

Returns: Returns an index used to identify the data product

Method raise_issue(): Record issue in fdp object

Usage:

```
fdp$raise_issue(
  index,
  type,
  use_data_product,
  use_component,
  use_version,
  use_namespace,
  issue,
  severity
)
```

Arguments:

index a numeric index, used to identify each input and output in the fdp object

type a string specifying the type of issue (one of "data", "config", "script", "repo")

use_data_product a string specifying the name of the data product, used as output in the code_run

use_component a string specifying the name of the data product component, used as output in the code_run

use_version a string specifying the name of the data product version, used as output in the code_run

use_namespace a string specifying the namespace in which the data product resides, used as input in the code_run

`issue` a `string` containing a free text description of the `issue`

`severity` an `integer` specifying the severity of the `issue`

Returns: Returns an updated `fdp` object

Method `finalise_output_hash()`**:** Record file hash and update path name in `fdp` object

Usage:

```
fdp$finalise_output_hash(
  use_data_product,
  use_data_product_runid,
  use_version,
  use_namespace,
  hash,
  new_path,
  data_product_url,
  delete_if_duplicate = FALSE
)
```

Arguments:

`use_data_product` a `string` specifying the name of the data product, used as output in the `code_run`

`use_data_product_runid` a `string` specifying the name of the data product, the same as `use_data_product` excluding the RUN_ID variable

`use_version` a `string` specifying the name of the data product version, used as output in the `code_run`

`use_namespace` a `string` specifying the namespace in which the data product resides, used as input in the `code_run`

`hash` a `string` specifying the hash of the file

`new_path` a `string` specifying the updated location (filename is now the hash of the file) of the data product in the local data store

`data_product_url` a `string` specifying the URL of an `object` associated with the `data_product`

`delete_if_duplicate` (optional) default is `FALSE`

Returns: Returns an updated `fdp` object

Method `finalise_output_url()`**:** Record `data_product` and component URLs in `fdp` object

Usage:

```
fdp$finalise_output_url(
  use_data_product,
  use_component,
  use_version,
  use_namespace,
  component_url
)
```

Arguments:

`use_data_product` a `string` specifying the name of the data product, used as output in the `code_run`

use_component a string specifying the name of the data product component, used as output in the code_run

use_version a string specifying the name of the data product version, used as output in the code_run

use_namespace a string specifying the namespace in which the data product resides, used as input in the code_run

component_url a string specifying the URL of an entry in the object_component table

Returns: Returns an updated fdp object

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

---

fdp_resolve_read *fdp_resolve_read*

Description

fdp_resolve_read

Usage

```
fdp_resolve_read(this_read, yaml)
```

Arguments

this_read	this_read
yaml	user written config file

---

fdp_resolve_write *fdp_resolve_write*

Description

fdp_resolve_write

Usage

```
fdp_resolve_write(this_write, yaml)
```

Arguments

this_write	this_write
yaml	user written config file

finalise *Finalise code run*

Description

Finalise Code Run and push associated metadata to the local registry.

Usage

```
finalise(handle, delete_if_empty = FALSE, delete_if_duplicate = FALSE)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
delete_if_empty	(optional) default is FALSE; see Details
delete_if_duplicate	(optional) default is FALSE; see Details

Details

If a Code Run does not read an input, write an output, or attach an issue, then delete the Code Run entry when delete_if_empty is set to TRUE.

If a data product has the same hash as a previous version, remove it from the registry when delete_if_duplicate is set to TRUE.

findme *findme*

Description

Returns metadata associated with the calculated hash of a target file. When multiple entries exist in the data registry all are returned.

Usage

```
findme(file, endpoint)
```

Arguments

file	file path
endpoint	endpoint

find_read_match *Find matching read aliases in config file*

Description

Find read aliases in working config that match wildcard string

Usage

```
find_read_match(handle, data_product)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the data product name

find_write_match *Find matching write aliases in config file*

Description

Find write aliases in working config that match wildcard string

Usage

```
find_write_match(handle, data_product)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the data product name

get_author_url *get_author_url*

Description

get_author_url

Usage

```
get_author_url(endpoint)
```

Arguments

endpoint	a string specifying the registry endpoint

get_components *Get H5 file components*

Description

Returns the names of the items at the root of the file

Usage

```
get_components(filename)
```

Arguments

filename	a string specifying a filename

Value

Returns the names of the items at the root of the file

See Also

Other get functions: get_entry(), get_existing(), get_file_hash(), get_github_hash()

get_dataproduct *get_dataproduct*

Description

get_dataproduct

Usage

```
get_dataproduct(
  data_product,
  version,
  namespace,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

data_product	data_product
version	version
namespace	namespace
endpoint	endpoint

get_entity *Get entity from url*

Description

Get entity from url

Usage

```
get_entity(url)
```

Arguments

url	a string specifying the url of an entry

get_entry *Return all fields associated with a table entry in the data registry*

Description

Return all fields associated with a table entry in the data registry

Usage

```
get_entry(table, query, endpoint = "http://localhost:8000/api/")
```

Arguments

table	a string specifying the name of the table
query	a list containing a valid query for the table, *e.g.* list(field = value)
endpoint	a string specifying the registry endpoint

Value

Returns a list of fields present in the specified entry

See Also

Other get functions: get_components(), get_existing(), get_file_hash(), get_github_hash()

get_existing *Return all entries posted to a table in the data registry*

Description

Get entries (from the data registry) in a particular table

Usage

```
get_existing(
  table,
  limit_results = TRUE,
  detail = "all",
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

table	a string specifying the name of the table
limit_results	a boolean specifying whether or not to limit the results, default is TRUE
detail	a string specifying what level of detail to return; options are "all" for all details or "id" for just URL and IDs
endpoint	a string specifying the registry endpoint

Value

Returns a data.frame of entries in table, default is limited to 100 entries

See Also

Other get functions: get_components(), get_entry(), get_file_hash(), get_github_hash()

get_fields *Get fields from table*

Description

Use API endpoint to produce a list of fields for a table. Requires API key.

Usage

```
get_fields(table, endpoint = "http://localhost:8000/api/")
```

Arguments

table	a string specifying the name of the table
endpoint	a string specifying the registry endpoint

Value

Returns a data.frame of fields and their attributes set to "none"

---

get_file_hash *Calculate hash from file*

Description

Returns the SHA1 hash of a given file

Usage

```
get_file_hash(filename)
```

Arguments

filename	a string specifying a filename

See Also

Other get functions: get_components(), get_entry(), get_existing(), get_github_hash()

---

get_github_hash *Get current GitHub hash*

Description

Get the hash of the latest commit in the master branch of a particular repository. This function assumes git is installed and located in the System PATH.

Usage

```
get_github_hash(repo)
```

Arguments

repo	a string specifying the github username/repository

See Also

Other get functions: get_components(), get_entry(), get_existing(), get_file_hash()

get_id *Get ID*

Description

Retrieve IDs for particular entries or all entries in a table

Usage

```
get_id(table, query = list(), endpoint = "http://localhost:8000/api/")
```

Arguments

table	a string specifying the name of the table
query	a list containing a valid query for the table, *e.g.* list(field = value)
endpoint	a string specifying the registry endpoint

Value

Returns a string or list of strings specifying the URL or URLs of entries in a table

get_index *get_index*

Description

get_index

Usage

```
get_index(write, data_product)
```

Arguments

write	write
data_product	data_product

get_max_version *get_max_version*

Description

If entry doesn't exist in the registry, return version 0.0.0

Usage

```
get_max_version(data_product, namespace_id)
```

Arguments

data_product	data_product
namespace_id	namespace_id

get_storage_location *Get storage location from url*

Description

Get storage location entry

Usage

```
get_storage_location(location)
```

Arguments

location	the url of an entry in the storage_location table

Value

Returns a list of fields associated with the specified entry

get_token *get_token*

Description

get_token

Usage

```
get_token()
```

get_url *Get URL*

Description

Retrieve URLs for particular entries or all entries in a table

Usage

```
get_url(table, query = list(), endpoint = "http://localhost:8000/api/")
```

Arguments

table	a string specifying the name of the table
query	a list containing a valid query for the table, *e.g.* list(field = value)
endpoint	a string specifying the registry endpoint

Value

Returns a string or list of strings specifying the URL or URLs of entries in a table

increment_filename *increment_filename*

Description

Searches directory for duplicate files and increments filename.

Usage

```
increment_filename(path)
```

Arguments

path	path

initialise *Initialise code run*

Description

Reads in a working config file, generates new Code Run entry, and returns a handle containing various metadata.

Usage

```
initialise(config, script)
```

Arguments

config	a string specifying the location of the working config file in the data store
script	a string specifying the location of the submission script in the data store

Value

Returns an object of class fdp,R6 containing metadata required by the Data Pipeline API

is_queryable *Check whether fields are queryable*

Description

Check whether fields are queryable

Usage

```
is_queryable(table, query, method, endpoint)
```

Arguments

table	a string specifying the name of the table
query	a list containing the query
method	a string specifying the method, c("GET", "POST")
endpoint	endpoint

Value

Returns TRUE if the entry is queryable and FALSE if it isn't

link_read *Link path to external format data*

Description

Link path to external format data

Usage

```
link_read(handle, data_product)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string representing an external object in the config.yaml file

Value

Returns a string specifying the location of the data product to be read

link_write *Link path for external format data*

Description

Link path for external format data

Usage

```
link_write(handle, data_product)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string representing an external object in the config.yaml file

Value

Returns a string specifying the location in which the data product should be written

new_author *Post entry to author table*

Description

Upload information to the author table in the data registry

Usage

```
new_author(name, identifier, endpoint = "http://localhost:8000/api/")
```

Arguments

name	a string specifying the full name or organisation name of the author; note that at least one of name or identifier must be specified
identifier	(optional) a string specifying the full URL identifier (*e.g.* ORCiD or ROR ID) of the author
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_code_repo_release *Post entry to code_repo_release table*

Description

Upload information to the code_repo_release table in the data registry

Usage

```
new_code_repo_release(
  name,
  version,
  object_url,
  website,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

name	a string specifying the name of an official release of code
version	a string specifying the version release (conforming with semantic versioning syntax)
object_url	a string specifying the URL of an object
website	(optional) a string specifying the URL of the website for this code release
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

---

new_code_run *Post entry to code_run table*

Description

Upload information to the code_run table in the data registry

Usage

```
new_code_run(
  run_date,
  description,
  code_repo_url,
  model_config_url,
  submission_script_url,
  inputs_urls = list(),
  outputs_urls = list(),
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

run_date	the date-time of the code_run *e.g.* Sys.time() or "2010-07-11 12:15:00 BST"
description	(optional) a string containing a free text description of the code_run
code_repo_url	(optional) a string specifying the URL of an object associated with the code_repo_release that was run
model_config_url	(optional) a string specifying the URL of an object associated with the working config file used for the code_run

submission_script_url	(optional) a string specifying the URL of an object associated with the submission script used for the code_run
inputs_urls	a list of object_component URLs referencing code_run inputs
outputs_urls	a list of object_component URLs referencing code_run outputs
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

---

new_data_product *Post entry to data_product table*

Description

Upload information to the data_product table in the data registry

Usage

```
new_data_product(
  name,
  version,
  object_url,
  namespace_url,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

name	a string specifying the name of the data_product
version	a string specifying the version identifier of the data_product (must conform to semantic versioning syntax)
object_url	a string specifying the URL of the entry in the object table
namespace_url	a string specifying the URL of the entry in the namespace table
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_external_object *Post entry to external_object table*

Description

Upload information to the external_object table in the data registry

Usage

```
new_external_object(
  doi_or_unique_name,
  primary_not_supplement = TRUE,
  release_date,
  title,
  description,
  data_product_url,
  original_store_url,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

doi_or_unique_name	a string specifying the DOI or name of the external_object
primary_not_supplement	(optional) a boolean flag to indicate whether the external object is the primary source (TRUE) or not (FALSE)
release_date	the date-time that the external_object was released *e.g.* Sys.time() or "2010-07-11 12:15:00 BST"
title	a string specifying the title of the external_object
description	(optional) a string containing a free text description of the external_object
data_product_url	a string specifying the URL of an entry in the data_product table
original_store_url	(optional) a string specifying the URL of a an entry in the storage_location table that references the original location of an external_object
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_file_type *Post entry to file_type table*

Description

Upload information to the file_type table in the data registry

Usage

```
new_file_type(name, extension, endpoint = "http://localhost:8000/api/")
```

Arguments

name	a string specifying the name of the file type
extension	a string specifying the filename extension
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_issue *Post entry to issue table*

Description

Upload information to the issue table in the data registry

Usage

```
new_issue(
  severity,
  description,
  component_issues,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

severity	an integer specifying the severity of the issue
description	a string containing a free text description of the issue
component_issues	a list of object_component URLs with which the issue is associated; this can be an empty list
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

---

new_keyword *Post entry to keyword table*

Description

Upload information to the keyword table in the data registry

Usage

```
new_keyword(
  object_url,
  keyphrase,
  identifier,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

object_url	a string specifying the URL of an object
keyphrase	a string a string containing a free text key phrase
identifier	(optional) a string specifying the URL of ontology annotation to associate with this keyword
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_licence *Post entry to licence table*

Description

Upload information to the licence table in the data registry

Usage

```
new_licence(object_url, licence_info, endpoint = "http://localhost:8000/api/")
```

Arguments

object_url	a string specifying the URL of an object
licence_info	a free text string containing information about the licence
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_namespace *Post entry to namespace table*

Description

Upload information to the namespace table in the data registry

Usage

```
new_namespace(
  name,
  full_name,
  website,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

name	a string specifying the name of the namespace
full_name	(optional) a string specifying the full name of the namespace
website	(optional) a string specifying the website URL associated with the namespace
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

---

new_object *Post entry to object table*

Description

Upload information to the object table in the data registry

Usage

```
new_object(
  description,
  storage_location_url,
  authors_url,
  file_type_url,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

description	(optional) a string containing a free text description of the object
storage_location_url	(optional) a string specifying the URL of an entry in the storage_location table
authors_url	(optional) a list of author URLs associated with this object
file_type_url	(optional) a string specifying the URL of an entry in the file_type table
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_object_component *Post entry to object_component table*

Description

Upload information to the object_component table in the data registry

Usage

```
new_object_component(
  object_url,
  name,
  description,
  whole_object = FALSE,
  issues_urls,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

object_url	a string specifying the URL of an existing object
name	a string specifying the name of the object_component, unique in the context of object_component and its object reference
description	(optional) a string containing a free text description of the object_component
whole_object	a boolean flag specifying whether or not this object_component refers to the whole object or not - default is FALSE
issues_urls	(optional) a list of issues URLs to associate with this object
endpoint	a string specifying the registry endpoint Note that the object_component table contains issues as an additional optional field. This is not included here. Instead use attach_issue() and associated functionality to attach issues to objects and object components.

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root(), new_user_author()

new_quality_controlled *Post entry to quality_controlled table*

Description

Upload information to the quality_controlled table in the data registry

Usage

```
new_quality_controlled(object_url, endpoint = "http://localhost:8000/api/")
```

Arguments

object_url	a string specifying the URL of an object
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_storage_location(), new_storage_root(), new_user_author()

new_storage_location *Post entry to storage_location table*

Description

Upload information to the storage_location table in the data registry

Usage

```
new_storage_location(
  path,
  hash,
  public,
  storage_root_url,
  endpoint = "http://localhost:8000/api/"
)
```

Arguments

path	a string specifying the path from the storage_root URI to the item location, which when appended to storage_root URI produces a complete URL
hash	a string specifying the SHA1 hash of a file stored in storage_location
public	a boolean indicating whether the storage_location is public or not (default is TRUE)
storage_root_url	a string specifying the URL of an entry in the storage_root table
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_root(), new_user_author()

---

new_storage_root *Post entry to storage_root table*

Description

Upload information to the storage_root table in the data registry

Usage

```
new_storage_root(root, local, endpoint = "http://localhost:8000/api/")
```

Arguments

root	a string specifying the URI of a storage_location, which when prepended to a storage_location produces a complete URI to a file
local	(optional) a boolean indicating whether the storage_root is local or not
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_user_author()

new_user_author *Post entry to user_author table*

Description

Upload information to the user_author table in the data registry

Usage

```
new_user_author(user_url, author_url, endpoint = "http://localhost:8000/api/")
```

Arguments

user_url	a string specifying the URL of an existing user
author_url	a string specifying the URL of an existing author
endpoint	a string specifying the registry endpoint

See Also

Other new functions: new_author(), new_code_repo_release(), new_code_run(), new_data_product(), new_external_object(), new_file_type(), new_issue(), new_keyword(), new_licence(), new_namespace(), new_object_component(), new_object(), new_quality_controlled(), new_storage_location(), new_storage_root()

paper_exists *Check whether paper exists*

Description

Check whether paper is in the data registry

Usage

```
paper_exists(doi)
```

Arguments

doi	doi

raise_issue *raise_issue*

Description

raise_issue

Usage

```
raise_issue(
  index,
  handle,
  component = NA,
  data_product,
  issue,
  severity,
  whole_object = FALSE
)
```

Arguments

index	index returned from link_*(), read_(), or write()
handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
component	a string specifying the component name
data_product	a string specifying the data product name
issue	a string specifying the issue
severity	a numeric value specifying the severity
whole_object	a boolean flag specifying whether or not to reference the whole_object

raise_issue_config *Raise issue with config file*

Description

Raise issue with config file

Usage

```
raise_issue_config(handle, issue, severity)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
issue	a string specifying the issue
severity	a numeric value specifying the severity

raise_issue_repo *Raise issue with remote repository*

Description

Raise issue with remote repository

Usage

```
raise_issue_repo(handle, issue, severity)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
issue	a string specifying the issue
severity	a numeric value specifying the severity

raise_issue_script *Raise issue with submission script*

Description

Raise issue with submission script

Usage

```
raise_issue_script(handle, issue, severity)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
issue	a string specifying the issue
severity	a numeric value specifying the severity

random_hash *random_hash*

Description

Generates a random hash

Usage

```
random_hash()
```

read_array *Read array component from HDF5 file*

Description

Function to read array type data from hdf5 file.

Usage

```
read_array(handle, data_product, component)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying a data product
component	a string specifying a data product component

Value

Returns an array with attached Dimension_i_title, Dimension_i_units, Dimension_i_values, and units attributes, if available

read_distribution *Read distribution component from TOML file*

Description

Function to read distribution type data from toml file.

Usage

```
read_distribution(handle, data_product, component)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying a data product
component	a string specifying a data product component

read_estimate *Read estimate component from TOML file*

Description

Function to read point-estimate type data from toml file.

Usage

```
read_estimate(handle, data_product, component)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying a data product
component	a string specifying a data product component

read_table *Read table component from HDF5 file*

Description

Function to read table type data from hdf5 file.

Usage

```
read_table(handle, data_product, component)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying a data product
component	a string specifying a data product component

Value

Returns a data.frame with attached column_units attributes, if available

register_issue_dataproduct *register_issue_dataproduct*

Description

register_issue_dataproduct

Usage

```
register_issue_dataproduct(handle, this_issue)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
this_issue	this_issue

register_issue_script *register_issue_script*

Description

register_issue_script

Usage

```
register_issue_script(handle, this_issue, type)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
this_issue	this_issue
type	type

`remove_empty_parents` *remove_empty_parents*

Description

remove_empty_parents

Usage

```
remove_empty_parents(path, root)
```

Arguments

`path`	path
`root`	root

`resolve_read` *resolve_read*

Description

resolve_read

Usage

```
resolve_read(handle, data_product, component = NA)
```

Arguments

`handle`	an object of class `fdp,R6` containing metadata required by the Data Pipeline API
`data_product`	a `string` specifying the name of the data product
`component`	a `string` specifying the name of data product component

resolve_version *resolve_version*

Description

resolve_version

Usage

```
resolve_version(version, data_product, namespace_id)
```

Arguments

version	version number
data_product	data_product
namespace_id	namespace_id

resolve_write *resolve_data_product*

Description

resolve_data_product

Usage

```
resolve_write(handle, data_product, file_type)
```

Arguments

handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the name of the data product
file_type	(optional) a string specifying the file type; when missing, file_type will be read from the config file

validate_fields *Validate fields*

Description

Function to validate fields in post data

Usage

```
validate_fields(table, data, endpoint)
```

Arguments

table	table name as character
data	data as a named list
endpoint	endpoint

Value

Returns

write_array *Write array component to HDF5 file*

Description

Function to populate hdf5 file with array type data.

Usage

```
write_array(
  array,
  handle,
  data_product,
  component,
  description,
  dimension_names,
  dimension_values,
  dimension_units,
  units
)
```

Arguments

array	an array containing the data
handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the name of the data product
component	a string specifying a location within the hdf5 file
description	a string describing the data product component
dimension_names	a list where each element is a vector containing the labels associated with a particular dimension (e.g. element 1 corresponds to dimension 1, which corresponds to row names) and the name of each element describes the contents of each dimension (e.g. age classes).
dimension_values	(optional) a list of values corresponding to each dimension (e.g. list element 2 corresponds to columns)
dimension_units	(optional) a list of units corresponding to each dimension (e.g. list element 2 corresponds to columns)
units	(optional) a string specifying the units of the data as a whole

Value

Returns a handle index associated with the just written component, which can be used to raise an issue if necessary

See Also

Other write functions: write_distribution(), write_estimate(), write_table()

---

write_distribution *Write distribution component to TOML file*

Description

Write distribution component to TOML file

Usage

```
write_distribution(
  distribution,
  parameters,
  handle,
  data_product,
  component,
  description
)
```

Arguments

distribution	a string specifying the name of the distribution
parameters	a list specifying the distribution parameters
handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the name of the data product
component	a string specifying a location within the toml file
description	a string describing the data product component

Value

Returns a handle index associated with the just written component, which can be used to raise an issue if necessary

See Also

Other write functions: write_array(), write_estimate(), write_table()

---

write_estimate *Write estimate component to TOML file*

Description

Function to populate toml file with point-estimate type data. If a file already exists at the specified location, an additional component will be added.

Usage

```
write_estimate(value, handle, data_product, component, description)
```

Arguments

value	an object of class numeric
handle	an object of class fdp,R6 containing metadata required by the Data Pipeline API
data_product	a string specifying the name of the data product
component	a string specifying a location within the toml file
description	a string describing the data product component

Value

Returns a handle index associated with the just written component, which can be used to raise an issue if necessary

See Also

Other write functions: write_array(), write_distribution(), write_table()

`write_table` *Write table component to HDF5 file*

Description

Function to populate hdf5 file with array type data.

Usage

```
write_table(
  df,
  handle,
  data_product,
  component,
  description,
  row_names,
  column_units
)
```

Arguments

`df`	an `dataframe` containing the data
`handle`	an object of class `fdp,R6` containing metadata required by the Data Pipeline API
`data_product`	a `string` specifying the name of the data product
`component`	a `string` specifying a location within the hdf5 file,
`description`	a `string` describing the data product component
`row_names`	(optional) a `vector` of rownames
`column_units`	(optional) a `vector` comprising column units

Value

Returns a handle index associated with the just written component, which can be used to raise an issue if necessary

See Also

Other write functions: `write_array()`, `write_distribution()`, `write_estimate()`

Index