Package: LEEF (via r-universe)

September 6, 2024

Type Package

Title Data Package Containing Only Data and Data Information

Version 0.9.1

Maintainer Rainer M. Krug <Rainer.Krug@uzh.ch>

BugReports https://github.com/LEEF-UZH/LEEF/issues

URL https://github.com/LEEF-UZH/LEEF

Description Setup package for the LEEF pipeline which loads / installs all necessary packages and functions to run the pipeline.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Additional_repositories https://leef-uzh.github.io/drat/

Depends R (>= 3.5.0),

Imports LEEF.measurement.bemovi, LEEF.measurement.flowcam, LEEF.measurement.flowcytometer, LEEF.measurement.manualcount, LEEF.measurement.o2meter, LEEF.archive.default, LEEF.backend.sqlite, testthat, yaml, drat, utils, R.utils, tools

Suggests covr, knitr, rmarkdown, shiny, shinyFiles

RoxygenNote 7.1.1

VignetteBuilder knitr

Repository https://leef-uzh.r-universe.dev

RemoteUrl https://github.com/LEEF-UZH/LEEF

RemoteRef master

RemoteSha 178c83e223ca3f0ff33cea2b4e9bb9f1d3ff626f

Contents

---

add *Add or replace a function in a queue*

Description

Functions starting with `add_...` do add a function to a queue which is processed by the coresponding `run_...` command.

The functions do reqquire exactly two arguments with the first named `input` and the second one named `output`. They should return either `TRUE` when run successful, or `FALSE` when failed. Although, the checking is not yet implemented.

Usage

```
add(fun, funname, queue)

add_additor(fun)

add_archiver(fun)

add_extractor(fun)

add_pre_processor(fun)
```

Arguments

`fun`	function which is run when calling `run_...()` The functions must not require any arguments!
`funname`	name of the function
`queue`	name of queue in `getOption("LEEF")`

Details

add: The function which is doing the adding - normally the specific `add_*` functions are used

add_additor: Adding a named function to the queue of additors. If the named function already exists will it be replaced.

add_archiver: Adding a named function to the queue of archivers. If the named function already exists will it be replaced.

These functions do archive the results of the current processing step.

add_extractor: Adding a named function to the queue of extractors. If the named function already exists will it be replaced.

These functions should extract data from the pre-processed data. The extracted data should be usable for the actual analysis to address the actual research question.

add_pre_processor: Adding a named function to the queue of pre-processors. If the named function already exists will it be replaced.

This function should pre-process the raw data. The pre-processed data should be archive ready, i.e. contain the same information as the raw data, be in an open format, and be compressed if possible.

Value

invisibly the function queue.

invisibly the function queue. A `list` which is processed

invisibly the function queue. A `list` which is processed

Examples

```
## Not run:
## To add the function `cat` to the `additor` queue
add (fun = cat, .queue = "additor")

## To add the function `paste` to the `extractor` queue
add (cat, "cat", "extractor")

## End(Not run)
add_additor( fun = cat )
add_additor( fun = cat )
add_extractor( fun = paste )
add_pre_processor( fun = paste )
```

`control_center` *Run the Control Center shiny app included in the LEEF package.*

Description

The Control centre app allows the

- sanity checks of the raw data
- running of the pipeline

Usage

```
control_center(rootdir = ".")
```

Arguments

`rootdir`	Directory in which all the data directories can be found.

Value

return value from `runApp()`

Examples

```
## Not run:
control_center()

## End(Not run)
```

`init_LEEF` *Create folder structure and prepares the pipeline based on the config file*

Description

The following steps are done in this function

Usage

```
init_LEEF(
  config_file = system.file("default_config.yml", package = "LEEF"),
  id = NULL
)
```

Arguments

`config_file`	config file to use. If none is specified, `cofig.yml` in the current working directory will be used.
`id`	id which will be appended to the name in the config file, using a '.'

Details

1. the config file as specified in the argument `config_file` is read
2. the folders as specified in the config file are, if they do not exist yet, created. If they are not specified, the following default values are used:
   - **general.parameter**: `00.general.parameter` - the directory containing general configuratuion files which are used for multiple measurements

- **raw**: `0.raw.data` - the raw data
- **pre_processed**: `1.pre_processed.data` - the pre-processed Archive Ready Data
- **extracted**: `2.extracted.data` = the extracted Research Ready Data
- **archive**: `3.archived.data` - the archived data from any of the previous steps or raw data
- **backend**: `9.backend` - the backend which contains the Research Ready Data from all pipeline runs before
- **tools**: `tools` - tools needed for running the different processes in the pipeline

3. verifies if a file named `sample_metadata.yml` exists which contains the metadata of the raw data
4. registers all `measurement`, `archive` and `backend` packages
5. verifies,if all tools are installed and installs them when needed. **THis step is specific to the bemovi measurement!!!**

Value

invisible `TRUE`

Examples

```
## Not run:
init_LEEF(system.file("default_config.yml", package = "LEEF"))

## End(Not run)
```

`LEEF` *LEEF*

Description

Meta package for LEEF pipeline

`list_LEEF_packages` *List packages on which LEEF depends from the LEEF-UZH repo*

Description

This function is a wrapper around `tools::package_dependencies("LEEF",which = "all", recursive = TRUE)` which returns only the packages which contain **.LEEF** or **LEEF.** and the package **LEEF** itself.

Usage

```
list_LEEF_packages(recursive = TRUE, versions = FALSE)
```

Arguments

`recursive`	logical: should (reverse) dependencies of (reverse) dependencies (and so on) be included? defaults to `TRUE`
`versions`	logical: should versions be returned as well.

Details

This function is a convenience function and only returns useful results when all packages which are dependencies of the `LEEF` package are prefixed with `LEEF.` or postficxed with `.LEEF`.

Value

list of all packages which are installed which contain **.LEEF** or **LEEF.** and the package **LEEF** itself

Examples

```
## Not run:
list_LEEF_packages()

## End(Not run)
```

`opt_directories` *Functions to read and write options*

Description

Read or write the directories to be used in the processing. Directories do not have to exist and will be created. Content will be overwritten without confirmation! If no parameter is given, the directories will be returned a a list.

Usage

```
opt_directories(
  general.parameter,
  raw,
  pre_processed,
  extracted,
  archive,
  tools
)
```

Arguments

`general.parameter`	`character vector` of length one containing the directory for the general parameter files
`raw`	`character vector` of length one containing the directory for the raw data

`pre_processed`	`character` `vector` of length one containing the directory for the pre_processed data
`extracted`	`character` `vector` of length one containing the directory for the extracted data
`archive`	`character` `vector` of length one containing the directory for the archived data
`tools`	directory in which the tools are located

Value

list of directories. If values have set, the value before the change.

Examples

```
opt_directories()

opt_directories(raw = "./temp")
```

`process` *Process all ques in the correct order*

Description

This function is an example and can be used as a template for processing the queues in a script,. Raw data is always archived using the "none" compression.

Usage

```
process(submitter, timestamp, process = TRUE, ...)
```

Arguments

`submitter`	name of submitter. When provided, will override the one in the 'sample_metadata.yml' file.
`timestamp`	timestamp for the data. When provided, will override the one in the 'sample_metadata.yml' file.
`process`	if `TRUE`, the pipeline will be processed. if `FALSE`, only the checks of the config file will be done nd no actual processing is happening.
`...`	additional arguments for the different queues

Value

invisibly `TRUE`

Examples

```
## Not run:
 process()

## End(Not run)
```

`process_raw_comp_none` *Process all queues in the correct order*

Description

This function is an example and can be used as a template for processing the queues in a script. It uses the archiver "none" for the raw and pre-processed data, useful for already compressed and large data, e.g. bemovi.

Usage

```
process_raw_comp_none(submitter, timestamp, ...)
```

Arguments

`submitter`	name of the submitter of the data to the pipeline. Will be added to the metadata.
`timestamp`	timestamp of the submission of the data to the pipeline. This should be in the format `YYYYMMDD` and will be used to identify the sampling day.
`...`	additional arguments for the different queues

Value

invisibly `TRUE`

Examples

```
## Not run:
 process()

## End(Not run)
```

`register_packages` *Register the functions to be usedfrom packages in the config file*

Description

Register the functions to be usedfrom packages in the config file

Usage

```
register_packages(packages)
```

Arguments

`packages`	list of packages. Each element **must** contain the elements `name` the name of the package, `InstallCommand` the command to be executed to install the package, and `RegisterCommand` the command to be executed to register the functions in a queue

Value

invisibly a list containing the results of the register commands

Examples

```
## Not run:
 register_packages(getOption("LEEF")$measurement_packages)

## End(Not run)
```

`run` *Run process queue*

Description

Run all the functions in the process queue named queue

Usage

```
run(input, output, queue)
```

Arguments

`input`	directory containing the input data in folders with the name of the methodology (e.g. `bemovi`)
`output`	directory in which the results will be written in a folder with the name of the methodology (e.g. `bemovi`)
`queue`	name of queue in `getOption("LEEF")`

Value

returns the results of the queue as a vector of length of the queue. If an element is `TRUE`, the function was run successfully (i.e. returned `TRUE`)

Examples

```
## Not run:
run(
  input = "./input",
  output = "./output",
  queue = "extractor"
)

## End(Not run)
```

`run_additors` *Run additors queue*

Description

Run all the additors registered with `add_additor()`.

Usage

```
run_additors()
```

Value

returns the results of the queue as a vector of length of the queue. If an element is `TRUE`, the function was run successfully (i.e. returned `TRUE`)

Examples

```
## Not run:
run_additors()

## End(Not run)
```

`run_archivers` *Run archivers queue*

Description

Run all the archivers registered with `add_archiver()`.

Usage

```
run_archivers(input, output)
```

Arguments

`input`	directory to be archive, including subdirectories
`output`	director in which the archive will be created

Value

returns the results of the queue as a vector of length of the queue. If an element is `TRUE`, the function was run successfully (i.e. returned `TRUE`)

Examples

```
## Not run:
run_archivers(
  input = "./input",
  output = "./output"
)

## End(Not run)
```

`run_extractors` *Run extractors queue*

Description

Run all the extractors registered with `add_extractor()`.

Usage

```
run_extractors()
```

Value

returns the results of the queue as a vector of length of the queue. If an element is `TRUE`, the function was run successfully (i.e. returned `TRUE`)

Examples

```
## Not run:
run_extractors()

## End(Not run)
```

`run_pre_processors` *Run pre_processors queue*

Description

Run all the additors registered with `add_pre_processor()`.

Usage

```
run_pre_processors()
```

Value

returns the results of the queue as a vector of length of the queue. If an element is `TRUE`, the function was run successfully (i.e. returned `TRUE`)

Examples

```
## Not run:
run_pre_processors()

## End(Not run)
```

`split_bemovi` *Split bemovi filder into a number of bemovi. folders with a maximum of per_batch video files*

Description

Split bemovi filder into a number of bemovi. folders with a maximum of per_batch video files

Usage

```
split_bemovi(
  per_batch = 30,
  bemovi_dir = file.path(".", "0.raw.data"),
  overwrite = TRUE
)
```

Arguments

`per_batch`	maximum number of movies per batch
`bemovi_dir`	bas directory in which the `bemovi` directory is located
`overwrite`	if `TRUE`, all folders starting with `bemovi.` in the `bemovi_dir` will be deleted

Value

the maximum id used

Examples

```
## Not run:
split_bemovi(per_batch = 5)

## End(Not run)
```

Index