Package 'rfacts'

July 23, 2025

Title R Interface to 'FACTS' on Unix-Like Systems

Description The 'rfacts' package is an R interface to the Fixed and Adaptive Clinical Trial Simulator ('FACTS') on Unix-like systems. It programmatically invokes 'FACTS' to run clinical trial simulations, and it aggregates simulation output data into tidy data frames. These capabilities provide end-to-end automation for large-scale simulation pipelines, and they enhance computational reproducibility. For more information on 'FACTS' itself, please visit <https://www.berryconsultants.com/software/>.

Version 0.2.1

License MIT + file LICENSE

URL https://elilillyco.github.io/rfacts/, https://github.com/EliLillyCo/rfacts

BugReports https://github.com/EliLillyCo/rfacts/issues

SystemRequirements FACTS Linux engines (>= 6.2.4), FLFLL (>= 6.2.4), Mono (>= 5.20.1.19)

Depends R (>= 3.6.0)

Imports digest (>= 0.6.25), fs (>= 1.3.1), tibble (>= 2.1.3), utils, xml2 (>= 1.2.2)

Suggests dplyr (>= 0.8.4), knitr (>= 1.28), rmarkdown (>= 2.1), testthat (>= 3.0.0), withr (>= 2.2.0)

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author William Michael Landau [aut, cre] (ORCID: <https://orcid.org/0000-0003-1878-3253>), Eli Lilly and Company [cph]

Maintainer William Michael Landau <will.landau@gmail.com>

Repository CRAN

Date/Publication 2022-08-19 13:10:02 UTC

Contents

`rfacts-package` *rfacts: interface to FACTS on Unix-like systems*

Description

Call FACTS from R.

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts") # example FACTS file
out <- run_facts(
  facts_file,
  n_sims = 4,
  verbose = FALSE
)
```

```
# What results files do we have?
head(get_csv_files(out))
# Read all the "patients*.csv" files with `read_patients(out)`.
# For each scenario, we have files named
# patients00001.csv, patients00002.csv, patients00003.csv,
# and patients00004.csv.
read_patients(out)
}
```

`facts_engines` *Engine-specific trial simulation functions*

Description

These functions are the inner functions called by `run_engine()`. In this help file, only the most common engine functions are listed. To identify the appropriate engine function for your FACTS file, call `get_facts_engine()`.

- `run_engine_aipf_contin()`: Enrichment continuous.
- `run_engine_aipf_dichot()`: Enrichment dichotomous.
- `run_engine_aipf_tte()`: Enrichment time to event.
- `run_engine_contin()`: Core continuous.
- `run_engine_crm()`: continual reassessment method (CRM).
- `run_engine_dichot()`: Core dichotomous.
- `run_engine_multep()`: Multiple endpoint.
- `run_engine_tte()`: Time to event.

Usage

```
run_engine_aipf_contin(
  param_files,
  n_sims = 1L,
  mode = c("", "r"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  interim = NULL,
  mcmc_num = NULL,
  verbose = FALSE,
  version = NULL
)

run_engine_aipf_dichot(
```

```
  param_files,
  n_sims = 1L,
  mode = c("", "r"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  interim = NULL,
  mcmc_num = NULL,
  verbose = FALSE,
  version = NULL
)

run_engine_aipf_tte(
  param_files,
  n_sims = 1L,
  mode = c("", "r"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  interim = NULL,
  mcmc_num = NULL,
  verbose = FALSE,
  version = NULL
)

run_engine_contin(
  param_files,
  n_sims = 1L,
  mode = c("s", "r", "p"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  arm_selection = NULL,
  armsdropped = NULL,
  complete_data_analysis = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  fsimdata = NULL,
  fsimexp = NULL,
  fsimparam = NULL,
  interim = NULL,
```

```
  keepfiles = NULL,
  mcmc_num = NULL,
  noadapt = NULL,
  s2_aux_paramfile = NULL,
  stage = NULL,
  verbose = FALSE,
  version = NULL
)

run_engine_crm(
  param_files,
  n_sims = 1L,
  mode = c("s", ""),
  directory = ".",
  allocator = NULL,
  charting_info = NULL,
  estimator = NULL,
  force_cohort = NULL,
  reduced_priority = NULL,
  version = NULL,
  verbose = FALSE
)

run_engine_dichot(
  param_files,
  n_sims = 1L,
  mode = c("s", "r", "p"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  arm_selection = NULL,
  armsdropped = NULL,
  complete_data_analysis = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  fsimdata = NULL,
  fsimexp = NULL,
  fsimparam = NULL,
  interim = NULL,
  keepfiles = NULL,
  mcmc_num = NULL,
  noadapt = NULL,
  s2_aux_paramfile = NULL,
  stage = NULL,
  verbose = FALSE,
  version = NULL
)
```

```
run_engine_multep(
  param_files,
  n_sims = 1L,
  mode = c("s", "r", "p"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  arm_selection = NULL,
  armsdropped = NULL,
  complete_data_analysis = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  fsimdata = NULL,
  fsimexp = NULL,
  fsimparam = NULL,
  interim = NULL,
  keepfiles = NULL,
  mcmc_num = NULL,
  noadapt = NULL,
  s2_aux_paramfile = NULL,
  stage = NULL,
  verbose = FALSE,
  version = NULL
)

run_engine_tte(
  param_files,
  n_sims = 1L,
  mode = c("s", "r", "p"),
  seed = NULL,
  analysis_data = NULL,
  analysis_mode = NULL,
  arm_selection = NULL,
  armsdropped = NULL,
  complete_data_analysis = NULL,
  current_week = NULL,
  execdata = NULL,
  final = NULL,
  fsimdata = NULL,
  fsimexp = NULL,
  fsimparam = NULL,
  interim = NULL,
  keepfiles = NULL,
  mcmc_num = NULL,
  noadapt = NULL,
  s2_aux_paramfile = NULL,
```

```
  stage = NULL,
  verbose = FALSE,
  version = NULL
)
```

Arguments

param_files	Character vector of file paths or the output of prep_param_files(). If a character vector, the elements can be directories containing *.param files or the paths to the *.param files themselves. Such a directory is returned by run_flfll().
n_sims	Positive integer, number of simulations per param file.
mode	Character scalar: "s" for simulation mode in non-enrichment designs, "" for simulation mode in enrichment designs, "r" for execution mode, and "p" for prediction mode. For the CRM engine, mode needs to be "s" or "".
seed	Positive integer, random number generator seed for the actual trial simulations. Use this seed argument instead of flfll_seed (run_facts(), run_flfll()) to control pseudo-randomness in the actual trial simulations. flfll_seed only controls how the *.param files are generated.
analysis_data	Character, analysis mode patient data file name.
analysis_mode	Logical, whether to activate analysis mode.
current_week	Numeric, current time in weeks.
execdata	Character, name of the execution mode patient file.
final	Logical, whether to do the final analysis. For execution mode only.
interim	Integer, interim number.
mcmc_num	Integer, MCMC file number. For analysis mode only.
verbose	Logical, whether to print progress information to the R console.
version	Character scalar, version of FACTS corresponding to the FACTS file. Get by calling get_facts_version() on your FACTS file. See possible versions with get_facts_versions(). Do not supply version to run_engine(). run_engine() detects the version automatically from the FACTS file and passes it to the appropriate engine function.
arm_selection	Logical, whether to activate arm selection.
armsdropped	Character, a comma-separated collection of integers indicating dropped arms.
complete_data_analysis	Logical, whether to do a complete data analysis.
fsimdata	Character, prediction mode patient data file name.
fsimexp	Logical. For expert use only.
fsimparam	Character, name of the prediction mode *.param file.
keepfiles	Logical, whether to deactivate cleanup of extraneous staged design files.
noadapt	Logical, whether to deactivate adaptive actions in prediction mode.
s2_aux_paramfile	Character, name of the stage 2 execution auxiliary *.param file.

stage	Integer, trial design stage. For staged designs only.
directory	Character, working directory. CRM only.
allocator	Logical, allocator/execution/recommender mode. CRM only.
charting_info	Logical, unused.
estimator	Logical, use estimator. CRM only.
force_cohort	Logical, whether to force small cohort run-in to end. CRM only.
reduced_priority	Logical, whether to run at reduced priority. CRM only.

Details

If you need to repeatedly invoke an engine, as with most trial execution mode workflows, these engine functions may be slow on their own. To avoid the most severe sources of slowness, consider running `prep_param_files()` and then passing the result to one of the individual engine functions (such as `run_engine_contin()`).

Value

Nothing.

See Also

`run_engine()`, `get_facts_file_example()`, `get_facts_engine()`, `run_facts()`, `run_flfll()`.

Examples

```
facts_file <- get_facts_file_example("contin.facts")
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
out <- run_flfll(facts_file, verbose = FALSE) # Generate param files.
# Identify which engine you need.
get_facts_engine(facts_file)
# Run the sims with the engine function or `run_engine()`.
run_engine_contin(out, n_sims = 1, verbose = FALSE, version = "6.2.5")
read_patients(out)
}
```

---

facts_results *Read trial simulation results*

Description

These functions read trial simulation results. The results were computed by FACTS (via `run_facts()` or `run_engine()` or one of the engine functions such as `run_engine_contin()`) and are stored in CSV files. Different functions read different types of output. The functions are named according to the CSV files they read. For example, `read_patients()` reads all files named `patients00001.csv`, `patients00002.csv`, etc. The most important functions are `read_patients()` and `read_weeks()`. The `read_s1*()`, `read_s2*()`, and `read_master*()` functions are for staged designs. The `read_csv_special()` function allows you to supply a custom file name prefix such as "patients", but be warned: not every kind of CSV output file is tested in `rfacts`.

Usage

```
read_patients(csv_files)

read_weeks(csv_files)

read_mcmc(csv_files)

read_s1_mcmc(csv_files)

read_s1_weeks(csv_files)

read_s1_patients(csv_files)

read_s2_patients(csv_files)

read_s2_weeks(csv_files)

read_s2_mcmc(csv_files)

read_master_mcmc(csv_files)

read_master_patients(csv_files)

read_master_weeks(csv_files)

read_cohorts(csv_files)

read_simulations(csv_files)

read_csv_special(csv_files, prefix, numbered = TRUE)
```

Arguments

`csv_files`	Character vector of file paths. Either the directories containing the trial simulation results or the actual CSV file files themselves.
`prefix`	Character, name of the prefix for `read_csv_special()`. `read_weeks(x)` is equivalent to `read_csv_special(x, prefix = "weeks")`. Be careful: not all

kinds of CSV output are tested. We can only guarantee the file types with special functions will be read correctly, e.g. `read_patients()` and `read_weeks()`.

`numbered` Logical. If `TRUE`, only read the numbered files like `patients00001.csv`, `weeks00017.csv`, etc. If `FALSE`, only list the non-numbered files like `simulations.csv` and `simulations_freq_locf.csv`. Avoid `summary.csv` files. They are not reliable on Linux.

Value

A data frame of trial simulation data. Each `read_*()` function returns different information, but all the `read_*()` functions support the following columns:

- `facts_file`: character, the base name of the FACTS file.
- `facts_scenario`: character, the name of the simulation scenario from FACTS. Usually, this factors in the virtual subject response (VSR) profile, accrual profile (how fast do patients enroll?) and dropout profile (how fast do they drop out?).
- `facts_sim`: integer, numeric index of the CSV file name. For example, the `facts_sim` of `patients00012.csv` is 12. In trial execution mode, all these indices are 00000, so `facts_id` is much safer than `facts_sim` for packetized trial execution mode.
- `facts_id`: character, random unique id of each CSV file being read. Different for every call to `read_patients()` etc. Safer than `facts_sim` for aggregation over simulations.
- `facts_output`: character, type of output is in the data frame: `"patients"` for patients files, `"weeks"` for weeks files, `"mcmc"` for MCMC files, etc. These names adhere to established conventions in FACTS.
- `facts_csv`: character, full path to the original CSV files where FACTS stored the simulation output. Required for `overwrite_csv_files()`.
- `facts_header`: a character vector of `\n`-delimited CSV file headers. Required for `overwrite_csv_files()`.

See Also

`get_facts_file_example()`, `run_facts()`, `run_flfll()`, `run_engine()`

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
out <- run_facts(
  facts_file,
  n_sims = 4,
  verbose = FALSE
)
# What results files do we have?
head(get_csv_files(out))
# Read all the "patients*.csv" files with `read_patients(out)`.
# For each scenario, we have files named
# patients00001.csv, patients00002.csv, patients00003.csv,
# and patients00004.csv.
read_patients(out)
}
```

get_csv_files *List FACTS-generated CSV files*

Description

List output CSV files in a directory or directories.

Usage

```
get_csv_files(csv_files, numbered = TRUE)
```

Arguments

csv_files	Character vector of directories containing numbered CSV files
numbered	Logical. If TRUE, only list the numbered files like patients00001.csv, weeks00017.csv, etc. If FALSE, only list the non-numbered files like simulations.csv and simulations_freq_locf.csv. Avoid summary.csv files. They are not reliable on Linux.

Value

A character vector of names of CSV files.

Examples

```
facts_file <- get_facts_file_example("contin.facts")
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
out <- run_facts(
  facts_file,
  n_sims = 2L,
  verbose = FALSE
)
get_csv_files(out)
}
```

get_facts_engine *Get the FACTS engine function matching your FACTS file*

Description

Identify the correct run_engine_*() function for your FACTS file.

Usage

```
get_facts_engine(facts_file)
```

Arguments

facts_file	Character, name of a FACTS file. Usually has a *.facts file extension.

Details

For most cases, it is sufficient to call run_facts(), or to call run_flfll() followed by run_engine(). But either way, you will need to know the arguments of the run_engine_*() function that corresponds to your FACTS file. Even if you are not calling this run_engine_*() directly, you will need to pass the arguments to ... in run_facts() or run_engine(). get_facts_engine() identifies the correct run_engine_*() function so you can open the help file and read about the arguments, e.g. ?run_engine_contin.

Value

Character, the name of a FACTS engine function.

See Also

run_facts(), run_engine()

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
out <- run_flfll(facts_file, verbose = FALSE) # Generate param files.
# Find the appropriate FACTS engine function.
get_facts_engine(facts_file)
# Read about the function arguments.
# You can pass these arguments to `...` in `run_facts()`
# or `run_engine()` or just call `run_engine_contin()` directly.
# ?run_engine_contin
# Call the FACTS engine function to run simulations.
# Alternatively, you could just call `run_engine()`.
run_engine_contin(out, n_sims = 1, verbose = FALSE, version = "6.2.5")
# See the results.
read_patients(out)
}
```

---

get_facts_file_example

Locate an example FACTS file

Description

Get the path to an example FACTS file inside rfacts itself.

Usage

```
get_facts_file_example(facts_file)
```

Arguments

facts_file	Character, name of a FACTS file. Usually has a `*.facts` file extension. Does not include the directory name. Possible choices: • "aipf_contin.facts" - Enrichment continuous. • "aipf_dichot.facts" - Enrichment dichotomous. • "aipf_tte.facts" - Enrichment time to event. • "broken.facts" - A broken FACTS file. • "contin.facts" - Core continuous. • "crm.facts" - N-CRM design. • "dichot.facts" - Core dichotomous. • "multep.facts" - Multiple endpoints. • "staged.facts" - Staged design. • "tte.facts" - Time to event. • "unsupported.facts" - FACTS file with an unsupported engine type.

Details

The `rfacts` package comes with some example FACTS files. Use the `get_facts_file_example()` function to get the full path to an example FACTS file. Use this file to try out `run_flfll()`, `run_engine_contin()`, etc.

Value

Character, the path to a FACTS file included with `rfacts`.

See Also

`run_facts()`, `run_flfll()`, `run_engine()`, `run_engine_contin()`

Examples

```
# Only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
facts_file
out <- run_facts(
  facts_file,
  n_sims = 1,
  verbose = FALSE
)
read_patients(out)
}
```

get_facts_scenarios *List the names of simulation scenarios*

Description

Get the names of the simulation scenarios of a FACTS file. without actually running any simulations. These names usually come from the virtual subject response (VSR) scenarios, the accrual profiles, and the dropout profiles.

Usage

```
get_facts_scenarios(facts_file, verbose = FALSE)
```

Arguments

facts_file	Character, name of a FACTS file. Usually has a *.facts file extension.
verbose	Logical, whether to print progress to the R console.

Value

Character vector of FACTS simulation scenarios.

See Also

get_param_dirs(), run_facts(), run_flfll(), run_engine(), run_engine_contin()

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
get_facts_scenarios(facts_file)
}
```

get_facts_version *Get FACTS version matching your FACTS file*

Description

Get the version of FACTS compatible with your *.facts file.

Usage

```
get_facts_version(facts_file)
```

Arguments

facts_file	Character, name of a FACTS file. Usually has a *.facts file extension.

Value

A version string.

See Also

get_facts_versions()

Examples

```
facts_file <- get_facts_file_example("contin.facts")
facts_file
get_facts_version(facts_file)
```

---

get_facts_versions *List supported FACTS versions*

Description

List versions of FACTS supported by rfacts. You can supply any of these versions to functions engine-specific functions such as run_engine_contin().

Usage

```
get_facts_versions()
```

Details

If your FACTS file does not perfectly agree with one of the supported versions, rfacts will try to find the best version for you, either

1. The greatest supported version less than or equal to the one in the FACTS file, or
2. The lowest supported version if (1) does not exist.

Value

A character vector of supported FACTS versions.

See Also

get_facts_version(), run_engine_contin()

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
get_facts_versions()
}
```

`get_param_dirs` *List the directories containing param files*

Description

Get the directory paths containing param files. This helps us run FACTS simulation scenarios one at a time.

Usage

```
get_param_dirs(param_files)
```

Arguments

`param_files`	Character, path to a top-level directory containing param files. `run_flfll()` and `run_facts()` return paths you can supply to `param_files` in `get_param_dirs()`.

Details

When you run `run_flfll()` or `run_facts()`, `rfacts` creates a directory. This directory has a bunch of subdirectories, each corresponding to a single simulation scenario (VSR profile x accrual profile x dropout profile, etc).

Value

Character vector of FACTS simulation scenario directories.

See Also

`get_facts_scenarios()`, `run_facts()`, `run_flfll()`, `run_engine()`, `run_engine_contin()`

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
param_files <- run_flfll(facts_file, verbose = FALSE)
scenarios <- get_param_dirs(param_files)
scenarios
scenario <- scenarios[1]
run_engine_contin(scenario, n_sims = 2, verbose = FALSE, version = "6.2.5")
read_patients(scenario)
}
```

get_param_files *List the paths to the param files*

Description

List the paths to the all the param files in a directory or directories.

Usage

```
get_param_files(param_files)
```

Arguments

param_files	Character vector of directories containing param files.

Value

Character vector of paths to param files.

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
dir <- run_flfll(facts_file, verbose = FALSE)
get_param_files(dir)
}
```

overwrite_csv_files *Overwrite FACTS CSV output files*

Description

read_patients() and friends read CSV output files from FACTS and return special aggregated data frames. overwite_csv_files() accepts such an aggregated data frame and writes the content to the original CSV files it came from.

Usage

```
overwrite_csv_files(x)
```

Arguments

x	An aggregated data frame from read_patients() or similar function.

Value

Nothing.

Examples

```
facts_file <- get_facts_file_example("contin.facts")
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
out <- run_facts(facts_file, n_sims = 2)
pats <- read_patients(out)
head(pats$visit_1)
pats$visit_1 <- 0
overwrite_csv_files(pats)
pats2 <- read_patients(out)
head(pats2$visit_1)
}
```

---

prep_param_files *Arrange the param files for the engines ahead of time.*

Description

If you call `prep_param_files()` ahead of time, subsequent calls to the engines will initialize much faster. This is useful in situations like trial execution mode, which require calling an engine function on each new simulation. This function does not actually modify the param files themselves on disk.

Usage

```
prep_param_files(param_files)
```

Arguments

`param_files`	A character vector of param files and/or directories containing param files.

Details

`prep_param_files()` searches for the required `*.param` files groups them by directory, sorts them, and returns the result as a list of special `param_files` objects. (It does not modify the actual contents of the `*.param`` files.) This preprocessing step is fast when executed once, but slow when executed th

Value

A list of special `"params_files"` objects that the engine functions can process fast.

See Also

`run_flfll()`, `run_engine()`, `run_engine_contin()`

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts")
out <- run_flfll(facts_file, verbose = FALSE)
param_files <- prep_param_files(out) # For speed.
param_files # Shows where the param files live and how they are organized.
run_engine_contin(
  param_files,
  n_sims = 2,
  verbose = FALSE,
  version = "6.2.5"
)
# Slower: run_engine_contin(out, n_sims = 2, verbose = FALSE) # nolint
}
```

`read_facts` *Read parts of FACTS files.*

Description

Read specific fields of a FACTS file.

Usage

```
read_facts(facts_file, fields)
```

Arguments

`facts_file`	Character of length 1, path to FACTS XML file to read.
`fields`	Data frame defining the kind of XML data to be read. It must have one row per field definition and the following columns: 1. `field`: custom name of the field. 2. `type`: value of the "type" attribute of the `<parameterSets>` tag. 3. `set`: value of the "name" attribute of the `<parameterSet>` tag. 4. `property`: value of the "name" attribute of the `<property>` tag.

Details

A FACTS file has a special kind of XML format. Most of the content sits in an overarching `<facts>` tag, then a `<parameterSets>` tag, then a `<parameterSet>` tag, then a `<property>` tag. For example, here is the part of a FACTS file that controls the weeks between interims.

```
<facts>
  <parameterSets type="NucleusParameterSet">
    <parameterSet name="nucleus">
      <property name="update_freq_save">4</property>
```

To use the `read_facts()` function, you must first identify the parts of the FACTS file you want to read using the `fields` argument. To read the above part of the XML, you would first define the `update_freq_save` field.

```
fields <- tibble::tibble(
  field = "my_interval",
  type = "NucleusParameterSet",
  set = "nucleus",
  property = "update_freq_save"
)
```

and then call `read_facts(input = "your_file.facts", fields = fields)`.

Value

A one-row `tibble` with the requested fields from the FACTS file.

Examples

```
facts_file <- get_facts_file_example("contin.facts")
fields <- data.frame(
  field = c("my_subjects", "my_vsr"),
  type = c("NucleusParameterSet", "EfficacyParameterSet"),
  set = c("nucleus", "resp2"),
  property = c("max_subjects", "true_endpoint_response")
)
read_facts(facts_file = facts_file, fields = fields)
```

---

reset_rfacts_paths *Reset system dependency info*

Description

Reset system dependency information based on the current value of the `RFACTS_PATHS` environment variable.

Usage

```
reset_rfacts_paths()
```

Dependencies

`rfacts` has strict system requirements, and the installations vary from system to system. You need to specify the locations of system executables in a CSV file that lists the path and metadata of each executable. This file must have one row per executable and the following columns.

- `executable_type`: Must be "mono", "flfll", or "engine" to denote the general type of the executable.

- `facts_version`: The version of FACTS with which this executable is compatible.
- `path`: File path to the executable.
- `engine_name`: For engines only. Name of the engine. Must be one of the engine types in the example CSV file at `system.file("example_paths.csv", package = "rfacts")`.
- `param_set`: For engines only. Parameter set designation listed in the XML code of FACTS files for that engine. See `system.file("example_paths.csv", package = "rfacts")` for examples.
- `param_type`: For engines only. Parameter type designation listed in the XML code of FACTS files for that engine. See `system.file("example_paths.csv", package = "rfacts")` for examples.

When you call a trial simulation function in `rfacts`, the package automatically reads this file and memorizes the contents for later use. The file at `system.file("example_paths.csv", package = "rfacts")` (`inst/example_paths.csv` in the package source.) has an example of such a file. All the columns in that file are required, and you may, remove, or modify rows to fit your specific system.

To enable `rfacts` to find this CSV file, you need to set the `RFACTS_PATHS` environment variable to the path to this file. The easiest way to do this is call `usethis::edit_r_environ()` to edit your `.Renviron` file and then add a new line with something like `RFACTS_PATHS=/path/to/file/paths.csv`. Then, restart your R session and call `Sys.getenv("RFACTS_PATHS")` to verify that this environment variable was set correctly.

The `rfacts_sitrep()` function inspects the current system dependency info and ensures each executable exists and has the correct permissions.

If you change the `RFACTS_PATHS` environment variable, you need to call `reset_rfacts_paths()` or restart R for the changes to take effect.

See Also

rfacts_paths, rfacts_sitrep

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
reset_rfacts_paths()
}
```

`rfacts_paths` *Read paths to rfacts system dependencies*

Description

Read the file specified by the `RFACTS_PATHS` environment variable.

Usage

```
rfacts_paths()
```

Value

A data frame with paths and other metadata about rfacts system dependencies

Dependencies

rfacts has strict system requirements, and the installations vary from system to system. You need to specify the locations of system executables in a CSV file that lists the path and metadata of each executable. This file must have one row per executable and the following columns.

- executable_type: Must be "mono", "flfll", or "engine" to denote the general type of the executable.
- facts_version: The version of FACTS with which this executable is compatible.
- path: File path to the executable.
- engine_name: For engines only. Name of the engine. Must be one of the engine types in the example CSV file at system.file("example_paths.csv", package = "rfacts").
- param_set: For engines only. Parameter set designation listed in the XML code of FACTS files for that engine. See system.file("example_paths.csv", package = "rfacts") for examples.
- param_type: For engines only. Parameter type designation listed in the XML code of FACTS files for that engine. See system.file("example_paths.csv", package = "rfacts") for examples.

When you call a trial simulation function in rfacts, the package automatically reads this file and memorizes the contents for later use. The file at system.file("example_paths.csv", package = "rfacts") (inst/example_paths.csv in the package source.) has an example of such a file. All the columns in that file are required, and you may, remove, or modify rows to fit your specific system.

To enable rfacts to find this CSV file, you need to set the RFACTS_PATHS environment variable to the path to this file. The easiest way to do this is call usethis::edit_r_environ() to edit your .Renviron file and then add a new line with something like RFACTS_PATHS=/path/to/file/paths.csv. Then, restart your R session and call Sys.getenv("RFACTS_PATHS") to verify that this environment variable was set correctly.

The rfacts_sitrep() function inspects the current system dependency info and ensures each executable exists and has the correct permissions.

If you change the RFACTS_PATHS environment variable, you need to call reset_rfacts_paths() or restart R for the changes to take effect.

See Also

rfacts_sitrep

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
rfacts_paths()
}
```

`rfacts_sitrep` *Check configuration of system dependencies*

Description

Examine the file paths to executables and check that they exist and have the correct permissions.

Usage

```
rfacts_sitrep()
```

Value

A data frame of information on the status of each executable.

Dependencies

`rfacts` has strict system requirements, and the installations vary from system to system. You need to specify the locations of system executables in a CSV file that lists the path and metadata of each executable. This file must have one row per executable and the following columns.

- `executable_type`: Must be "mono", "flfll", or "engine" to denote the general type of the executable.
- `facts_version`: The version of FACTS with which this executable is compatible.
- `path`: File path to the executable.
- `engine_name`: For engines only. Name of the engine. Must be one of the engine types in the example CSV file at `system.file("example_paths.csv", package = "rfacts")`.
- `param_set`: For engines only. Parameter set designation listed in the XML code of FACTS files for that engine. See `system.file("example_paths.csv", package = "rfacts")` for examples.
- `param_type`: For engines only. Parameter type designation listed in the XML code of FACTS files for that engine. See `system.file("example_paths.csv", package = "rfacts")` for examples.

When you call a trial simulation function in `rfacts`, the package automatically reads this file and memorizes the contents for later use. The file at `system.file("example_paths.csv", package = "rfacts")` (`inst/example_paths.csv` in the package source.) has an example of such a file. All the columns in that file are required, and you may, remove, or modify rows to fit your specific system.

To enable `rfacts` to find this CSV file, you need to set the `RFACTS_PATHS` environment variable to the path to this file. The easiest way to do this is call `usethis::edit_r_environ()` to edit your `.Renviron` file and then add a new line with something like `RFACTS_PATHS=/path/to/file/paths.csv`. Then, restart your R session and call `Sys.getenv("RFACTS_PATHS")` to verify that this environment variable was set correctly.

The `rfacts_sitrep()` function inspects the current system dependency info and ensures each executable exists and has the correct permissions.

If you change the RFACTS_PATHS environment variable, you need to call reset_rfacts_paths() or restart R for the changes to take effect.

See Also

rfacts_paths

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
rfacts_sitrep()
}
```

`run_engine` *Run trial simulations*

Description

For fine control over trial simulations, you must first call run_flfll() and then call either run_engine() or one of the specific engine functions (such as run_engine_contin()). The engines read the *.param files generated by run_flfll(), run the trial simulations, and save output to a bunch of CSV files. You can find these CSV output files next to the *.param files.

Usage

```
run_engine(facts_file, ...)
```

Arguments

facts_file	Character, name of a FACTS file. Usually has a *.facts file extension.
...	Named arguments to the appropriate inner engine function, such as run_engine_contin(). Use get_facts_engine() to identify the appropriate engine function for your FACTS file. Then, open the help file of that function to read about the arguments.

Details

If you need to repeatedly invoke an engine, as with most trial execution mode workflows, run_engine() is slow. Instead, consider running prep_param_files() and then passing the result to one of the individual engine functions (such as run_engine_contin()).

Value

Nothing.

See Also

get_facts_file_example(), run_flfll(), get_facts_engine(), prep_param_files()

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts") # example FACTS file
out <- run_flfll(facts_file, verbose = FALSE) # Generate param files.
# Run the simulations.
run_engine(
  facts_file,
  param_files = out,
  n_sims = 1,
  verbose = FALSE
)
read_patients(out)
}
```

`run_facts` *Run FACTS*

Description

Run FACTS trial simulations.

Usage

```
run_facts(
  facts_file,
  output_path = tempfile(),
  log_path = output_path,
  n_burn = NULL,
  n_mcmc = NULL,
  n_weeks_files = 10000,
  n_patients_files = 10000,
  n_mcmc_files = 0,
  n_mcmc_thin = NULL,
  flfll_seed = NULL,
  flfll_offset = NULL,
  n_sims,
  ...
)
```

Arguments

`facts_file`	Character, name of a FACTS file. Usually has a `*.facts` file extension.
`output_path`	Character, directory path to the files to generate.
`log_path`	Character, path to the log file generated by FLFLL.
`n_burn`	Number of burn-in iterations for the MCMC.
`n_mcmc`	Number of MCMC iterations used in inference.

n_weeks_files	Number of weeks*.csv files to save in output_path.
n_patients_files	Number of patients*.csv files to save in output_path.
n_mcmc_files	Number of mcmc*.csv files to save in output_path.
n_mcmc_thin	Number of thinning iterations for the MCMC.
flfll_seed	Positive integer, random number generator seed for FLFLL. This seed is only used for stochastic preprocessing steps for generating the *.param files. It is not the random number generator seed for the actual trial simulations. To set the trial simulation seed, use the seed argument of run_facts(), run_engine(), or one of the specific run_engine*() functions.
flfll_offset	Integer, offset for the random number generator.
n_sims	Positive integer, number of simulations per param file.
...	Named arguments to the appropriate FACTS engine function. Use get_facts_engine() to identify the appropriate engine function and then open the help file of that function to read about the arguments, e.g. ?run_engine_contin.

Details

run_facts() calls run_flfll() and then run_engine(). For finer control over trial simulation, you can call these latter two functions individually.

Value

Character, path to the directory with FACTS output.

See Also

run_flfll(), run_engine(), get_facts_engine()

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts") # example FACTS file
out <- run_facts(
  facts_file,
  n_sims = 4,
  verbose = FALSE
)
# What results files do we have?
head(get_csv_files(out))
# Read all the "patients*.csv" files with `read_patients(out)`.
# For each scenario, we have files named
# patients00001.csv, patients00002.csv, patients00003.csv,
# and patients00004.csv.
read_patients(out)
}
```

`run_flfll` *Generate param files to prepare for trial simulations*

Description

Generate the preparatory files required for simulation.

Usage

```
run_flfll(
  facts_file,
  output_path = tempfile(),
  log_path = output_path,
  n_burn = NULL,
  n_mcmc = NULL,
  n_weeks_files = 10000,
  n_patients_files = 10000,
  n_mcmc_files = 0,
  n_mcmc_thin = NULL,
  flfll_seed = NULL,
  flfll_offset = NULL,
  verbose = FALSE,
  max_sims = 99999L
)
```

Arguments

`facts_file`	Character, name of a FACTS file. Usually has a `*.facts` file extension.
`output_path`	Character, directory path to the files to generate.
`log_path`	Character, path to the log file generated by FLFLL.
`n_burn`	Number of burn-in iterations for the MCMC.
`n_mcmc`	Number of MCMC iterations used in inference.
`n_weeks_files`	Number of `weeks*.csv` files to save in `output_path`.
`n_patients_files`	Number of `patients*.csv` files to save in `output_path`.
`n_mcmc_files`	Number of `mcmc*.csv` files to save in `output_path`.
`n_mcmc_thin`	Number of thinning iterations for the MCMC.
`flfll_seed`	Positive integer, random number generator seed for FLFLL. This seed is only used for stochastic preprocessing steps for generating the `*.param` files. It is not the random number generator seed for the actual trial simulations. To set the trial simulation seed, use the `seed` argument of `run_facts()`, `run_engine()`, or one of the specific `run_engine*()` functions.
`flfll_offset`	Integer, offset for the random number generator.
`verbose`	Logical, whether to print progress information to the R console.

max_sims — Positive integer of length 1, maximum number of simulations that will be allowed to run for certain engines like CRM in subsequent calls to the engine. If the n_sims argument of the engine is larger than max_sims, only max_sims simulations will be run. The max_sims argument only applies to FLFLL >= 6.4.1 and only needs to be set manually if you are manually calling run_flfll() and then the engine instead of just run_facts().

Details

For advanced control over trial simulations, you must first call run_flfll() and then call one of the engine functions such as run_engine_contin(). run_flfll() generates the preparatory *.param files that the run_engine_*() functions understand. You will pass these *.param files or their parent directory to param_files argument of run_engine_contin() etc.

Value

Character, the value of output_path. output_path is the directory path to the files generated by run_flfll().

See Also

get_facts_file_example(), run_engine(), run_engine_contin()

Examples

```
# Can only run if system dependencies are configured:
if (file.exists(Sys.getenv("RFACTS_PATHS"))) {
facts_file <- get_facts_file_example("contin.facts") # example FACTS file
out <- run_flfll(facts_file, verbose = FALSE) # Generate param files.
# Run the simulations.
run_engine(
  facts_file,
  param_files = out,
  n_sims = 1,
  verbose = FALSE
)
read_patients(out)
}
```

---

write_facts — *Write modified FACTS files.*

Description

Write modified versions of existing FACTS files. This function can be used to tweak properties of a FACTS file such as maximum sample size, number of weeks between interims, allocation ratios, data generation parameters, and analysis priors.

Usage

```
write_facts(fields, values, default_dir = "_facts")
```

Arguments

`fields`	Data frame defining the kind of XML data to be replaced. It must have one row per field definition and the following columns: 1. `field`: custom name of the field. 2. `type`: value of the "type" attribute of the `<parameterSets>` tag. 3. `set`: value of the "name" attribute of the `<parameterSet>` tag. 4. `property`: value of the "name" attribute of the `<property>` tag.
`values`	Data frame defining the FACTS files to generate. Must have one row per FACTS file and a column called `facts_file` with the names of the input FACTS files. An `output` column with the names of the output FACTS files is recommended but not required. (If `output` is not specified, the output FACTS files will be written to automatically generated paths inside `default_dir`.) Other columns must have names corresponding to elements of `fields$field` and contain values to insert into the FACTS files. These columns could be vectors or lists of vectors. In the former case, each element is a scalar replacement to a property. In the latter case, an XML property receives an entire vector as an item list, and the vector must be the same length as the original item list.
`default_dir`	Directory to write the output FACTS files if `values` has no `output` column.

Details

A FACTS file has a special kind of XML format. Most of the content sits in an overarching `<facts>` tag, then a `<parameterSets>` tag, then a `<parameterSet>` tag, then a `<property>` tag. For example, here is the part of a FACTS file that controls the weeks between interims.

```
<facts>
  <parameterSets type="NucleusParameterSet">
    <parameterSet name="nucleus">
      <property name="update_freq_save">4</property>
```

To use the `write_facts()` function, you must first identify the parts of the FACTS file you want to modify (the `fields` argument) then the values that should be substituted in (the `values` argument). Given the XML above, to create new FACTS files with intervals 5 and 6 instead of 4, you would set

```
fields <- tibble::tibble(
  field = "my_interval",
  type = "NucleusParameterSet",
  set = "nucleus",
  property = "update_freq_save"
)
values <- tibble::tibble(
  facts_file = "your_facts_file.facts",
  output = "output_file.facts",
```

```
  my_interval = c(5, 6)
)
```

and then call `write_facts(fields = fields, values = values)`.

Value

The function writes FACTS XML files and returns a character vector with the paths to those files.

Examples

```
# Identify a source FACTS file.
facts_file <- get_facts_file_example("contin.facts")
# Create 4 new FACTS files with different numbers of max patients.
fields <- data.frame(
  field = "my_subjects",
  type = "NucleusParameterSet",
  set = "nucleus",
  property = "max_subjects"
)
values <- data.frame(
  facts_file = facts_file,
  output = c("_facts/out1000.facts", "_facts/out2000.facts"),
  my_subjects = c(1000, 2000)
)
default_dir <- tempfile()
write_facts(fields = fields, values = values, default_dir = default_dir)
list.files("_facts")
unlink("_facts", recursive = TRUE)
```

Index