Package ‘fz’

November 9, 2026

Type Package

Title R Wrapper for the 'funz-fz' Parametric Simulation Framework

Version 2.0.0

Description Provides R bindings to the 'funz-fz' Python package using 'reticulate'. The 'fz' framework wraps arbitrary simulation codes to run parameter sweeps, design-of-experiments studies, and iterative algorithm-driven analyses by substituting variable placeholders in text input files and collecting outputs into data frames. Calculators can run locally (shell), over SSH, or on 'SLURM' clusters.
See <https://github.com/Funz/fz> for the underlying framework.

License BSD_3_clause + file LICENSE

Encoding UTF-8

Language en-US

SystemRequirements Python (>= 3.9), funz-fz Python package

Imports reticulate (>= 1.41), utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/reticulate list(packages = list(list(package = ``funz-fz'')))

VignetteBuilder knitr

URL https://github.com/Funz/fz.R

BugReports https://github.com/Funz/fz.R/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Yann Richet [aut, cre] (ORCID: <https://orcid.org/0000-0002-5677-8458>)

Maintainer Yann Richet <yann.richet@asnr.fr>

Repository CRAN

Date/Publication 2026-11-08 23:00:08 UTC

Contents

fzc . . . 2
fzd . . . 3
fzi . . . 6
fzl . . . 7
fzo . . . 8
fzr . . . 9
fz_available . . . 11
fz_install . . . 12
get_config . . . 13
get_interpreter . . . 14
get_log_level . . . 14
install . . . 15
install_algorithm . . . 15
install_model . . . 16
list_installed_algorithms . . . 17
list_installed_models . . . 17
list_models . . . 18
print_config . . . 18
reload_config . . . 19
set_interpreter . . . 19
set_log_level . . . 20
uninstall . . . 21
uninstall_algorithm . . . 21
uninstall_model . . . 22

Index **23**

`fzc` *fzc Function*

Description

Compiles input file(s) by replacing variable placeholders with values. Each unique combination of values is written to its own subdirectory inside `output_dir`, named `var1=val1,var2=val2,...`.

Usage

```
fzc(
  input_path,
  input_variables,
  model,
  output_dir = "output",
  input_static = NULL
)
```

Arguments

input_path	Path to input file or directory.
input_variables	Named list of variable values. Supply a vector of values to generate a full-factorial grid across variables.
model	Model definition dict or alias string.
output_dir	Output directory for compiled files. Default "output".
input_static	Optional character vector of files that are identical across every case (see fzr's input_static). They are symlinked into output_dir rather than templated or duplicated. Default NULL.

Value

NULL (invisibly). Called for side effects.

Examples

```
if (fz_available()) {
  tf <- tempfile(fileext = ".txt")
  writeLines(c("P = ${P~1.013}", "V = ${V~22.4}"), tf)

  model <- list(varprefix = "$", delim = "{}", formulaprefix = "@",
                commentline = "#")
  out <- tempfile()

  fzc(tf, list(P = 2.0, V = 11.2), model, out)
  fzc(tf, list(P = c(1.0, 2.0), V = c(11.2, 22.4)), model, out)
}
```

---

fzd *fzd Function*

Description

Runs an iterative design of experiments driven by an algorithm. Unlike fzr (which evaluates a fixed grid), fzd lets an algorithm adaptively choose which parameter combinations to evaluate, which is useful for sensitivity analysis, surrogate-model fitting, or optimization.

Usage

```
fzd(
  input_path,
  input_variables,
  model,
  output_expression = NULL,
  algorithm,
```

```
  calculators = NULL,
  algorithm_options = NULL,
  analysis_dir = "analysis",
  input_static = NULL
)
```

Arguments

input_path	Path to input file or directory. Must be NULL when model is an R function (see "Direct function model" below).
input_variables	Named list of variable range strings of the form "[min;max]", e.g. list(x = "[0;1]", y = "[-5;5]").
model	Model definition dict or alias string, or an R function (see "Direct function model" below).
output_expression	Expression evaluated on the model outputs to produce the quantity the algorithm optimizes or analyses, e.g. "result" or "out1 + 2 * out2". Vector-valued outputs may be reduced with mean(), sum(), len(), median(), stdev(), variance(), indexing/slicing, and zip(). A character vector of length > 1 requests a multi-objective run (funz-fz >= 1.2): each case then yields one scalar per expression, passed as-is to multi-objective algorithms such as NSGA-II. May be NULL only when model is a function, in which case the first output value is used.
algorithm	Path to the algorithm Python file, e.g. "algorithms/montecarlo_uniform.py".
calculators	Calculator specification(s). Default NULL. When model is a function, this must be a single integer (default 1L) and is always forced to 1L (see "Direct function model" below): R functions are only safe to call from the main thread, so a value other than 1 triggers a warning and is overridden.
algorithm_options	Algorithm options as a named list, a JSON string, or a path to a JSON file. Default NULL.
analysis_dir	Analysis directory. Default "analysis".
input_static	Optional character vector of files identical across every case (see fzr's input_static); passed through unchanged to each iteration's internal fzr() call. Default NULL.

Value

Named list with the analysis results produced by the algorithm.

Direct function model

Instead of a file-based model, model can be an R function. This requires funz-fz >= 1.2 (earlier releases do not support callable models); a development build can be installed with fz_install(packages = "git+https://github.com/Funz/fz.git"). In this mode:

- input_path must be NULL – there are no input files.
- input_variables names must match the function's arguments.

- `output_expression` may be `NULL`; the value used is then the first element of the function's return value (its return value directly if scalar, the first element if a vector/list, or the first entry's value if a named list).
- `calculators` must be a single integer, but is always forced to 1 here – regardless of the value passed in – before being forwarded to `fz`. On the `fz` (Python) side, `calculators` > 1 now evaluates a Python-function model concurrently in a worker-thread pool (Funz/fz#73); that is unsafe here because R functions are called back into the R session via `reticulate`, which is only safe from the main thread – invoking the function from any other thread crashes the R session. Passing a value other than 1 therefore emits a warning explaining that it is being forced back to 1, and the call always proceeds with `calculators` = 1 (strictly sequential, one call at a time, in the calling thread).
- each iteration's directory (`iterNNN/`) only contains a `values.csv` of that iteration's function inputs/outputs, since there is no file-based execution.

Examples

```
if (fz_available()) {
  # run inside a throwaway directory: fz writes analysis/ and .fz/ under cwd
  ex_dir <- file.path(tempdir(), "fz-fzd-example")
  dir.create(ex_dir, showWarnings = FALSE)
  owd <- setwd(ex_dir)

  writeLines(c("x = ${x~0}", "y = ${y~0}"), "input.txt")

  # The calculator: reads the compiled input.txt and writes output.txt
  writeLines(
   "awk -F= '/^x/ {x = $2} /^y/ {y = $2} END {print \"z=\" x + y}' input.txt > output.txt",
   "compute.sh"
  )

  model <- list(
    varprefix = "$", delim = "{}", formulaprefix = "@", commentline = "#",
    output = list(z = "grep z output.txt | cut -d= -f2")
  )

  # A minimal self-contained random-sampling algorithm (see
  # https://github.com/Funz/fz for ready-made algorithms to install)
  algo <- tempfile(fileext = ".py")
  writeLines(c(
    "import random",
    "class RandomSampler:",
    "    def __init__(self, **options):",
    "        self.batch = int(options.get('batch_sample_size', 5))",
    "        self.max_iterations = int(options.get('max_iterations', 3))",
    "        self.iteration = 0",
    "        self.input_vars = {}",
    "    def get_initial_design(self, input_vars, output_vars):",
    "        self.input_vars = input_vars",
    "        self.iteration = 1",
    "        return [{k: random.uniform(*v) for k, v in input_vars.items()}",
    "                for _ in range(self.batch)]",
```

```
    "    def get_next_design(self, previous_input_vars, previous_output_values):",
    "        self.iteration += 1",
    "        if self.iteration > self.max_iterations:",
    "            return []",
    "        return [{k: random.uniform(*v) for k, v in self.input_vars.items()}",
    "                for _ in range(self.batch)]",
    "    def get_analysis(self, input_vars, output_values):",
    "        valid = [v for v in output_values if v is not None]",
    "        mean = sum(valid) / len(valid) if valid else None",
    "        return {'text': f'mean={mean}', 'data': {'mean': mean}}"
  ), algo)

  result <- fzd(
    "input.txt",
    list(x = "[0;1]", y = "[-5;5]"),
    model,
    output_expression = "z",
    algorithm        = algo,
    calculators      = "sh://bash compute.sh",
    algorithm_options = list(batch_sample_size = 10, max_iterations = 3)
  )

  setwd(owd)
  unlink(ex_dir, recursive = TRUE)
}


## Not run:
# Direct function model (requires funz-fz >= 1.2)
rosenbrock <- function(x, y) {
  list(result = (1 - x)^2 + 100 * (y - x^2)^2)
}

result <- fzd(
  input_path = NULL,
  input_variables = list(x = "[-2;2]", y = "[-2;2]"),
  model = rosenbrock,
  output_expression = "result",
  algorithm = "examples/algorithms/bfgs.py",
  calculators = 1L, # forced to 1L anyway for R functions -- see "Direct function model"
  algorithm_options = list(max_iter = 20, tol = 1e-4)
)

## End(Not run)
```

`fzi` *fzi Function*

Description

Parses input file(s) to find variables, formulas, and static objects.

Usage

```
fzi(input_path, model, input_static = NULL)
```

Arguments

input_path	Path to input file or directory.
model	Model definition dict or alias string.
input_static	Optional character vector of files that are identical across every case (see fzr's input_static). They are never scanned for variables, since they are never templated. Default NULL.

Value

Named list keyed by the discovered static objects, variable names, and formula expressions, mapped to their values (or NULL for variables with no default).

Examples

```
if (fz_available()) {
  tf <- tempfile(fileext = ".txt")
  writeLines(c("pressure = ${P~1.013}", "volume = ${V~22.4}"), tf)

  model <- list(varprefix = "$", delim = "{}", formulaprefix = "@",
                commentline = "#")

  vars <- fzi(tf, model)
}
```

---

fzl *fzl Function*

Description

Lists installed models and available calculators.

Usage

```
fzl(models = "*", calculators = "*", check = FALSE)
```

Arguments

models	Pattern to match models. Default "*" for all. Accepts glob patterns ("my*") or plain alias names.
calculators	Pattern to match calculators. Default "*" for all.
check	Logical; probe each calculator to verify it is reachable. Default FALSE.

Value

Named list with two entries:

models Named list of installed model definitions.

calculators Named list of available calculators.

Examples

```
if (fz_available()) {
  info <- fzl()
  names(info$models)
  names(info$calculators)

  info <- fzl(models = "Perfect*")
  info <- fzl(check = TRUE)
}
```

`fzo` *fzo Function*

Description

Reads and parses output file(s) according to the model's output commands. Each matched directory is processed independently; the results are combined into a single list or data frame.

Usage

```
fzo(output_path, model)
```

Arguments

`output_path`	Path or glob pattern matching one or more output directories. Subdirectories within matched directories are not processed.
`model`	Model definition dict or alias string.

Value

Named list or data frame of parsed output values. An output entry may resolve to a vector (list) - e.g. a time series or spectrum - which is stored per case unmodified (no flattening, padding, or truncation).

Output extraction methods

Each entry of the model's `output` list is a string describing how to extract that value from the case directory. Besides the default shell command (optionally marked `"bash://..."`), `funz-fz` >= 1.2 supports shell-free extractors, portable on Windows without a bash install:

- `"python://<expr>"` - a Python expression evaluated in the case directory with helpers such as `grep()`, `read()`, `lines()`, `json_file()`, `csv_file()`, `hdf5_file()` and the `re`/`json`/`math`/`statistics`/`np`/`pd` modules.
- `"jq://<filter> <file>"` - JSON extraction via the `jq` executable.
- `"yq://<filter> <file>"` - YAML/JSON/XML/TOML extraction via `mikefarah/yq`.
- `"xpath://<expr> <file>"` - XML extraction via `xmllint --xpath`.

The `python://`, `jq://`, `yq://` and `xpath://` forms preserve list results as vectors; a plain shell command simplifies a single-element result to a scalar.

Examples

```
if (fz_available()) {
  out_dir <- file.path(tempdir(), "P=2,V=11.2")
  dir.create(out_dir, recursive = TRUE)
  writeLines("result = 42", file.path(out_dir, "output.txt"))

  model <- list(
    varprefix = "$", delim = "{}", formulaprefix = "@", commentline = "#",
    output = list(result = "grep 'result' output.txt | cut -d= -f2")
  )

  values <- fzo(out_dir, model)
}
```

---

fzr *fzr Function*

Description

Runs full parametric calculations over an input template. fzr combines `fzc`, calculator execution, and `fzo` into a single call: it compiles the template for every parameter combination, runs the model via the calculator(s), and collects all outputs into a data frame.

Usage

```
fzr(
  input_path,
  input_variables,
  model,
  results_dir = "results",
```

```
  calculators = NULL,
  callbacks = NULL,
  timeout = NULL,
  case_naming = NULL,
  input_static = NULL
)
```

Arguments

input_path	Path to input file or directory.
input_variables	Named list of variable values (or vectors of values for a full-factorial grid), or a data frame where each row is one case.
model	Model definition dict or alias string.
results_dir	Results directory. Default "results".
calculators	Calculator specification(s). Strings of the form "sh://<command>" run a local shell command; "ssh://user\@host" runs over SSH; NULL auto-detects installed calculators.
callbacks	Optional named list of callback functions.
timeout	Timeout in seconds per case. Default NULL, which resolves to the model's own "timeout" entry if set, otherwise the FZ_RUN_TIMEOUT configuration value (1 hour by default for sh:// and funz://; since funz-fz 2.0, no timeout for ssh:// and slurm:// unless FZ_RUN_TIMEOUT is set). An explicit value here takes precedence over both. 0 means "no timeout" (funz-fz >= 2.0; earlier releases timed every case out immediately); a negative value is an error.
case_naming	How each case's result/temp subdirectory is named: "path" (default, "var1=val1,var2=val2,..."), "hash" (short content hash of the variable combination), or "index" ("case_<i>"). "hash"/"index" avoid filesystem name length limits with many variables and write a cases.csv manifest at the results root. Default NULL (uses FZ_CASE_NAMING or "path").
input_static	Optional character vector of files identical across every case (e.g. a shared reference dataset): never templated, never re-hashed per case, and - for relative paths - symlinked into each case directory instead of duplicated (and transferred to remote calculators). Absolute paths are assumed already present calculator-side. Default NULL.

Value

Data frame (or named list) with one row per case and columns for each input variable and output quantity.

Examples

```
if (fz_available()) {
  # run inside a throwaway directory: fz writes results/ and .fz/ under cwd
  ex_dir <- file.path(tempdir(), "fz-fzr-example")
  dir.create(ex_dir, showWarnings = FALSE)
```

```
  owd <- setwd(ex_dir)

  # The input file is itself the script run by the calculator below
  writeLines(c(
    "#!/bin/sh",
    "echo result = $(( ${x~0} + ${y~0} )) > output.txt"
  ), "input.sh")

  model <- list(
    varprefix = "$", delim = "{}", formulaprefix = "@", commentline = "#",
    output = list(result = "grep result output.txt | cut -d= -f2")
  )

  results <- fzr("input.sh", list(x = c(1L, 2L), y = 3L), model,
                 calculators = "sh://bash input.sh")

  setwd(owd)
  unlink(ex_dir, recursive = TRUE)
}
```

`fz_available` *Check if fz Python Package is Available*

Description

Checks whether the fz Python package is available in the current Python environment.

Usage

```
fz_available()
```

Details

The importable module name of the `funz-fz` package is `fz`. This function returns `TRUE` only when a module named `fz` imports *and* exposes the expected API (`fzi`/`fzc`/`fzo`/ `fzr`), so that an unrelated Python package that also happens to be importable as `fz` is not mistaken for `funz-fz`. All examples and tests are guarded with `if (fz_available())` and therefore skip (rather than error) when the Python side is missing or misconfigured.

Value

Logical; TRUE if fz is available, FALSE otherwise.

Version synchronisation

This package expects the `funz-fz` release with the same major and minor version as itself. When no Python is configured, **reticulate** provisions that version automatically in an ephemeral environment on first use, which downloads Python and `funz-fz` (see `reticulate::py_require()`). An

existing Python environment is used as is: if its `funz-fz` version differs, this function offers to install the matching one in an interactive session, and warns otherwise. Nothing is changed without consent. Set `options(fz.sync_python = FALSE)` to disable this check. Neither step runs during `R CMD check`.

Examples

```
if (fz_available()) {
  message("fz is available!")
} else {
  message("Please install fz with fz_install()")
}
```

`fz_install` *Install the fz Python Package*

Description

This function installs the fz Python package into a virtual environment or conda environment managed by reticulate.

Usage

```
fz_install(
  packages = fz_python_requirement(),
  method = "auto",
  conda = "auto",
  pip = TRUE,
  ...
)
```

Arguments

`packages`	Package specification passed to `reticulate::py_install()`. The default installs the `funz-fz` release series matching this R package (same major and minor version, e.g. `"funz-fz>=2.0,<2.1"` for R package 2.0.0). To track unreleased features, install the latest `main` branch directly from GitHub with `"git+https://github.com/Funz/fz.git"` and set `options(fz.sync_python = FALSE)` to silence the version check described in `fz_available()`.
`method`	Installation method. Either "auto", "virtualenv", or "conda".
`conda`	Path to conda executable. Only used when method is "conda".
`pip`	Logical; use pip for installation? Default is TRUE.
`...`	Additional arguments passed to `reticulate::py_install()`.

Value

NULL (invisibly). Called for side effects.

Examples

```
## Not run:
# Install fz in a virtual environment
fz_install()

# Install in a conda environment
fz_install(method = "conda")

# Track the latest main branch on GitHub (unreleased features)
fz_install(packages = "git+https://github.com/Funz/fz.git")

## End(Not run)
```

---

get_config *Get the Global Configuration*

Description

Returns the fz configuration object. Values are controlled by environment variables such as `FZ_LOG_LEVEL`, `FZ_MAX_WORKERS`, `FZ_MAX_RETRIES`, and `FZ_SHELL_PATH`.

Usage

```
get_config()
```

Value

A Python `Config` object. Access fields with `$`, e.g. `get_config()$max_workers`.

Examples

```
if (fz_available()) {
  cfg <- get_config()
  cfg$max_workers
  cfg$max_retries
}
```

get_interpreter *Get the Current Interpreter*

Description

Returns the global formula interpreter used when evaluating formula expressions inside template files (e.g. `"python"` or `"R"`).

Usage

```
get_interpreter()
```

Value

Character string naming the current interpreter.

Examples

```
if (fz_available()) {
  get_interpreter()
}
```

get_log_level *Get the Current Log Level*

Description

Returns the current logging verbosity level.

Usage

```
get_log_level()
```

Value

A log-level value (use `as.character()` to convert to a string such as `"DEBUG"`, `"INFO"`, `"WARNING"`, `"ERROR"`).

Examples

```
if (fz_available()) {
  as.character(get_log_level())
}
```

install *Install a Model or Algorithm (generic)*

Description

Generic alias: installs a model from a GitHub name, URL, or local zip file. Equivalent to `install_model`.

Usage

```
install(source, global = FALSE)
```

Arguments

source	GitHub name (e.g. `"Funz/Model-PerfectGas"`), URL, or path to a local zip file.
global	Logical; install system-wide instead of user-level. Default `FALSE`.

Value

Named list with installation details.

Examples

```
## Not run:
# Requires the named GitHub repository to exist and network access;
# not run automatically since neither is guaranteed in all environments.
install("Funz/Model-PerfectGas")

## End(Not run)
```

install_algorithm *Install an Algorithm*

Description

Installs an algorithm from a GitHub repository name, URL, or local zip file into the user-level `~/.fz/algorithms/` directory (or system-level when `global = TRUE`).

Usage

```
install_algorithm(source, global = FALSE)
```

Arguments

source	GitHub name (e.g. `"Funz/Algorithm-MonteCarlo"`), URL, or path to a local zip file.
global	Logical; install system-wide instead of user-level. Default `FALSE`.

Value

Named list with installation details (path, name, ...).

Examples

```
## Not run:
# Requires the named GitHub repository to exist and network access;
# not run automatically since neither is guaranteed in all environments.
install_algorithm("Funz/Algorithm-MonteCarlo")

## End(Not run)
```

`install_model` *Install a Model*

Description

Installs a model from a GitHub repository name, URL, or local zip file into the user-level `~/.fz/models/` directory (or system-level when `global = TRUE`).

Usage

```
install_model(source, global = FALSE)
```

Arguments

`source`	GitHub name (e.g. `"Funz/Model-PerfectGas"`), URL, or path to a local zip file.
`global`	Logical; install system-wide instead of user-level. Default `FALSE`.

Value

Named list with installation details (path, id, ...).

Examples

```
## Not run:
# Requires the named GitHub repository to exist and network access;
# not run automatically since neither is guaranteed in all environments.
install_model("Funz/Model-PerfectGas")

## End(Not run)
```

list_installed_algorithms *List Installed Algorithms*

Description

Returns details of all algorithms installed in `~/.fz/algorithms/`.

Usage

```
list_installed_algorithms(global = FALSE)
```

Arguments

`global`	Logical; list system-level installs. Default `FALSE`.

Value

Named list of installed algorithm definitions.

Examples

```
if (fz_available()) {
  algos <- list_installed_algorithms()
  names(algos)
}
```

list_installed_models *List Installed Models*

Description

Returns details of all models installed in `~/.fz/models/`.

Usage

```
list_installed_models(global = FALSE)
```

Arguments

`global`	Logical; list system-level installs. Default `FALSE`.

Value

Named list of installed model definitions.

Examples

```
if (fz_available()) {
  models <- list_installed_models()
  names(models)
}
```

`list_models` *List Installed Models (alias)*

Description

Alias for `list_installed_models`.

Usage

```
list_models(global = FALSE)
```

Arguments

`global`	Logical; list system-level installs. Default `FALSE`.

Value

Named list of installed model definitions.

Examples

```
if (fz_available()) {
  names(list_models())
}
```

`print_config` *Print the Current Configuration*

Description

Prints all fz configuration values in a human-readable format, including which settings come from environment variables.

Usage

```
print_config()
```

Value

NULL (invisibly). Called for side effects.

Examples

```
if (fz_available()) {
  print_config()
}
```

---

reload_config *Reload Configuration from Environment Variables*

Description

Re-reads all `FZ_*` environment variables and updates the live configuration. Useful after changing environment variables within the session.

Usage

```
reload_config()
```

Value

NULL (invisibly). Called for side effects.

Examples

```
if (fz_available()) {
  Sys.setenv(FZ_MAX_WORKERS = "8")
  reload_config()
  get_config()$max_workers
}
```

---

set_interpreter *Set the Interpreter*

Description

Sets the global formula interpreter for evaluating expressions inside template files.

Usage

```
set_interpreter(interpreter)
```

Arguments

interpreter	Character string: "python" or "R".

Value

NULL (invisibly). Called for side effects.

Examples

```
if (fz_available()) {
  set_interpreter("R")
  set_interpreter("python")
}
```

---

set_log_level *Set the Log Level*

Description

Controls how much output fz emits during execution.

Usage

```
set_log_level(level)
```

Arguments

level	Character string or log-level object: one of "DEBUG", "INFO", "WARNING", "ERROR".

Value

NULL (invisibly). Called for side effects.

Examples

```
if (fz_available()) {
  set_log_level("DEBUG")
  set_log_level("WARNING")
  set_log_level("ERROR")
}
```

uninstall *Uninstall a Model (generic)*

Description

Generic alias: removes a model by name. Equivalent to `uninstall_model`.

Usage

```
uninstall(model_name, global = FALSE)
```

Arguments

`model_name`	Name of the model to remove.
`global`	Logical; remove from system-level install. Default `FALSE`.

Value

`TRUE` if removed, `FALSE` otherwise.

Examples

```
if (fz_available()) {
  uninstall("PerfectGas")
}
```

uninstall_algorithm *Uninstall an Algorithm*

Description

Removes a previously installed algorithm from `~/.fz/algorithms/`.

Usage

```
uninstall_algorithm(algorithm_name, global = FALSE)
```

Arguments

`algorithm_name`	Name of the algorithm to remove.
`global`	Logical; remove from system-level install. Default `FALSE`.

Value

`TRUE` if the algorithm was removed, `FALSE` otherwise.

Examples

```
if (fz_available()) {
  uninstall_algorithm("MonteCarlo")
}
```

`uninstall_model` *Uninstall a Model*

Description

Removes a previously installed model from `~/.fz/models/`.

Usage

```
uninstall_model(model_name, global = FALSE)
```

Arguments

`model_name`	Name of the model to remove (e.g. `"PerfectGas"`).
`global`	Logical; remove from system-level install. Default `FALSE`.

Value

`TRUE` if the model was removed, `FALSE` otherwise.

Examples

```
if (fz_available()) {
  uninstall_model("PerfectGas")
}
```

Index

fz_available, 11
fz_available(), *12*
fz_install, 12
fzc, 2, *9*
fzd, 3
fzi, 6
fzl, 7
fzo, 8, *9*
fzr, *3*, *4*, *7*, 9

get_config, 13
get_interpreter, 14
get_log_level, 14

install, 15
install_algorithm, 15
install_model, *15*, 16

list_installed_algorithms, 17
list_installed_models, 17, *18*
list_models, 18

print_config, 18

reload_config, 19
reticulate::py_install(), *12*
reticulate::py_require(), *11*

set_interpreter, 19
set_log_level, 20

uninstall, 21
uninstall_algorithm, 21
uninstall_model, *21*, 22