Package: qspquest (via r-universe)

November 18, 2026

Type Package

Title Pharmacology Model Simulation and Bounded Estimation

Version 0.1.0

Description Compile pharmacokinetic, pharmacodynamic and quantitative systems pharmacology models with explicit physical units, then simulate dosing regimens, run parameter scans and Morris (1991) <doi:10.1080/00401706.1991.10484804> elementary-effects screens. Parameters are estimated within declared bounds by maximum likelihood or maximum a posteriori fitting for individual and pooled data, and by first-order conditional estimation with interaction (Wang 2007 <doi:10.1007/s10928-007-9060-6>) or stochastic approximation expectation-maximization (Kuhn and Lavielle 2005 <doi:10.1016/j.csda.2004.07.002>) for population data. Count and correlated binomial data can be fitted with Gamma-Poisson and probit-normal mixture models. A documented subset of 'NONMEM'-style dosing and observation records can be converted to model requests with source-row provenance. Computation uses the 'QSPQuest' engine, written in 'Rust', with stiff ordinary differential equation solvers from the 'diffsol' library.

License Apache License (== 2.0)

URL https://github.com/unibio-intelligence/qspquest, https://unibiointelligence.com, https://biologics.unibiointelligence.com/

BugReports https://github.com/unibio-intelligence/qspquest/issues

Encoding UTF-8

Imports jsonlite (>= 2.0.0)

Suggests rxode2

SystemRequirements Cargo (Rust's package manager; >= 1.89), rustc (>= 1.89)

NeedsCompilation yes

Copyright UniBio Intelligence; authors and copyright holders of the bundled Rust crates, listed in inst/COPYRIGHTS.

OS_type unix

Author Vivek Kohar [aut, cre], UniBio Intelligence [cph, fnd], Authors of the dependency Rust crates [ctb, cph] (see inst/COPYRIGHTS and inst/THIRD-PARTY-NOTICES.txt)

Maintainer Vivek Kohar <vivek.kohar@unibiointelligence.com>

Config/pak/sysreqs libclang-dev

Repository https://cran.r-universe.dev

Date/Publication 2026-11-17 18:50:08 UTC

RemoteUrl https://github.com/cran/qspquest

RemoteRef HEAD

RemoteSha 6c74f122139f74237814b93f0d8a8dfac5c15468

Contents

qspquest-package 2
compile_model 3
fit_data_requests 4
fit_gamma_poisson 6
fit_parameter_data_frame 7
model_library 8
nonmem_requests 9
population_fit_data_requests 11
qspquest_export_rxode2_ui 13
result_data_frame 14

Index **15**

`qspquest-package` *Unit-Aware Pharmacology Simulation and Bounded Estimation*

Description

Compile pharmacology models with explicit units and run simulation, parameter scans, sensitivity analysis and bounded estimation using a native Rust engine. Native simulation and fitting do not require Python or network services.

Details

A complete runnable tutorial is installed at `system.file("doc", "quickstart.R", package = "qspquest")`. Run it with `source()` after installation. A second example uses the synthetic model and request in the installed `extdata` directory. Always inspect fitting status and declared limitations before interpreting results.

See Also

compile_model, fit_gamma_poisson, fit_data_requests, population_fit_data_requests

---

compile_model *Compile and Execute Pharmacology Models*

Description

Compile a model once and execute versioned simulation, bounded fitting, parameter scan, or Morris sensitivity requests with the Rust engine.

Usage

```
compile_model(source)
simulate_model(model, request)
fit_model(model, request)
scan_model(model, request)
morris_model(model, request)
```

Arguments

source	A model document as an R list or JSON string, or a model text string.
model	The session-local compiled model returned by compile_model.
request	A versioned request as an R list or JSON string. Use the run, fit, scan, or Morris schema corresponding to the operation. Arrays should be R lists, including arrays containing just one item.

Details

Compiled pointers cannot be transferred to another process or restored from saved R data. Recompile the source in each process. Fitting compiles its selected sensitivities for each call. JSON models have a four-million-byte limit; model text and requests retain a one-million-byte limit. Units, bounds, event semantics and computational budgets are validated by the engine. Scientific errors are R conditions of class qspquest_error, with code, message and any available location fields. Fitting supports bounded individual and pooled Gaussian MLE/MAP with fixed event times and fixed residual-error parameters. Population random effects and censored likelihoods are not supported by these operations.

Value

compile_model returns a qspquest_model containing an external pointer. The execution functions return parsed, versioned result lists with execution identity and scientific values. Inspect resultfitstatus after fitting; a returned result does not imply convergence.

See Also

result_data_frame, nonmem_requests, fit_data_requests

Examples

```
source <- paste(
  "model synthetic_decay",
  "time_unit = h",
  "parameters",
  "  k = 0.1 [1/h] log",
  "states",
  "  amount = 6 [mg]",
  "dynamics",
  "  d/dt(amount) = -k * amount",
  "outputs",
  "  remaining = amount [mg]",
  "end", sep = "\n")
model <- compile_model(source)
quantity <- function(value, unit) list(value = value, unit = unit)
request <- list(
  schema = "qspquest.run/v0.1", solver = "diffsol_bdf",
  budgets = list(solver_callbacks = 100000L, events = 100L,
                 output_values = 1000L),
  regimen = list(end = quantity(24, "h"),
    samples = lapply(c(0, 6, 12, 24), quantity, unit = "h"),
    administrations = list(), rtol = 1e-9, atol = 1e-11))
result <- simulate_model(model, request)
trajectory <- result_data_frame(result)
stopifnot(max(abs(trajectory$VALUE - 6 * exp(-0.1 * trajectory$TIME))) < 1e-6)
trajectory
```

`fit_data_requests` *Prepare and Fit Gaussian Observation Datasets*

Description

Map measurement records to explicit model outputs, units and fixed Gaussian error models, then fit subjects independently or pool selected parameters.

Usage

```
fit_data_requests(dataset, template, endpoints, mode = c("individual", "pooled"))
fit_data(model, dataset)
```

Arguments

`dataset`	For preparation, a `qspquest_dataset` from `nonmem_requests`. For execution, a `qspquest_fit_dataset` from `fit_data_requests`.
`template`	An individual `qspquest.fit/v0.1` request list or JSON string supplying bounds, optimizer controls, priors and a sensitivity selection. Its objective observations and simulation run must be empty; the dataset supplies them.

endpoints	Named list keyed by observation CMT. Each entry contains exactly output, value_unit and error. The error contains a unit-bearing additive_sd and dimensionless proportional_sd.
mode	"individual" fits each subject independently; "pooled" shares selected parameters across subjects.
model	A compiled model from compile_model.

Details

Included DV values must be finite; use MDV to exclude missing measurements. Observation CMT requires an explicit endpoint mapping. Duplicate observations remain independent likelihood terms. SOURCE_ROW is one-based in the original data; RESULT_ROW is zero-based in that subject's native simulation. The engine converts compatible measurement/error units. A pooled MAP prior is counted once; independent fits each receive the template prior. Subjects' requests execute sequentially. This supports fixed-error Gaussian fits, without population random effects, censoring, residual-error estimation or inferred covariate mappings.

Value

Preparation returns inspectable requests, mapping, subject_ids, mode and original source. Execution returns the same mapping and source plus results and the inspectable versioned requests for optional prediction replay. Inspect every result's fit$status before using estimates.

See Also

nonmem_requests, fit_model, fit_parameter_data_frame

Examples

```
q <- function(value, unit) list(value = value, unit = unit)
model <- compile_model(paste(
  "model one_compartment", "time_unit = h", "parameters",
  "  cl = 0.6 [L/h] log", "  v = 4.5 [L] log", "states", "  central = 0 [mg]",
  "dynamics", "  d/dt(central) = input_central - cl / v * central",
  "outputs", "  cp = central / v [mg/L]", "dosing", "  central [mg] = 1",
  "end", sep = "\n"))
run_template <- list(schema = "qspquest.run/v0.1", solver = "diffsol_bdf",
  budgets = list(solver_callbacks = 1000000L, events = 100L,
                 output_values = 10000L),
  regimen = list(end = q(4, "h"), samples = list(), administrations = list(),
                 rtol = 1e-10, atol = 1e-12))
rows <- do.call(rbind, lapply(c("A", "B"), function(id) data.frame(
  ID = id, TIME = c(0, 1, 2, 4), EVID = c(1L, 0L, 0L, 0L),
  CMT = c(1L, 2L, 2L, 2L), AMT = c(6, 0, 0, 0), MDV = c(1L, 0L, 0L, 0L),
  DV = c(NA, 6 / 4.5 * exp(-0.6 * c(1, 2, 4) / 4.5)))))
dataset <- nonmem_requests(rows, run_template, c("1" = "central"), "h", "mg")
template <- list(schema = "qspquest.fit/v0.1", problem = list(
  kind = "individual", request = list(
    objective = list(simulation = list(schema = "qspquest.sensitivity/v0.1",
        with_respect_to = list("cl"),
        derivative_absolute_tolerances = list(cl = list(central = q(1e-12, "mg.h/L")))),
```

```
      observations = list(), priors = list()),
    bounds = list(list(parameter = "cl", lower = q(0.05, "L/h"),
                       upper = q(1.2, "L/h"))),
    max_evaluations = 400L, max_iterations = 100L, gradient_tolerance = 1e-5)))
endpoints <- list("2" = list(output = "cp", value_unit = "mg/L",
  error = list(additive_sd = q(0.1, "mg/L"), proportional_sd = 0)))
prepared <- fit_data_requests(dataset, template, endpoints, mode = "pooled")
fitted <- fit_data(model, prepared)
stopifnot(fitted$results[[1]]$fit$status == "converged")
fitted$results[[1]]$fit$parameters
head(prepared$mapping)
```

`fit_gamma_poisson` *Fit Count or Correlated Binomial Data*

Description

Bounded native point estimation for independent Gamma-Poisson count/exposure rows or two groups of independent binomial litters with normally distributed probit effects.

Usage

```
fit_gamma_poisson(request)
fit_probit_normal_binomial(request)
```

Arguments

`request`	A complete request as an R list or a JSON string, using the `qspquest.gamma-poisson-fit/v0.1` or `qspquest.probit-normal-binomial-fit/v0.1` schema, respectively.

Details

Gamma rates have exposure units in the shape-rate convention. Probit means and SDs are dimensionless, with optimization in mean and variance coordinates. The probit-normal mixture integrates over an unbounded normal distribution and supports zero SD. These functions calculate point estimates without confidence intervals.

Value

A list with the fit status, estimates, normalized negative log likelihood, work counts, projected gradient norm and source/request identities. Check `status` before interpreting estimates. Quadrature-budget exhaustion raises a structured `qspquest_error`.

Examples

```
q <- function(value) list(value = value, unit = "h")
request <- list(
  schema = "qspquest.gamma-poisson-fit/v0.1",
  observations = list(list(count = 0L, exposure = q(1)),
                      list(count = 1L, exposure = q(1))),
  initial_shape = 1, shape_bounds = c(0.01, 100),
  initial_rate = q(1), rate_bounds = list(q(0.01), q(100)),
  max_evaluations = 1000L, max_iterations = 200L,
  gradient_tolerance = 1e-7)
fit <- fit_gamma_poisson(request)
stopifnot(fit$status == "converged")
fit$parameters
```

`fit_parameter_data_frame` *Turn Dataset Fits Into Plot-Ready Tables*

Description

Extract unit-bearing parameter estimates or replay converged fits at their final estimates to obtain source-row-aligned observations and predictions.

Usage

```
fit_parameter_data_frame(fitted)
fit_observation_data_frame(model, fitted)
```

Arguments

`fitted`	A result returned by `fit_data`.
`model`	The compiled model used by `fit_data`.

Details

Prediction replay runs one simulation per subject on demand, using the fitted parameters and the original request. It requires every fit to have `status == "converged"`. Original subject-specific execution results remain available in the `fit_data` result. These helpers cover the individual and pooled fixed-error Gaussian dataset adapter; they do not infer random effects, covariates, censoring, or residual-error models.

Value

`fit_parameter_data_frame` returns ID, PARAMETER, ESTIMATE, UNIT, STATUS and OBJECTIVE columns. Pooled fits have a missing ID because estimates are shared. `fit_observation_data_frame` returns ID, SOURCE_ROW, RESULT_ROW, TIME, SIDE, OUTPUT, DV, DV_UNIT, PRED, PRED_UNIT and RESID. RESID is missing when DV and PRED use different units; the package never subtracts incompatible values. Row order matches the source observations within each subject.

See Also

fit_data_requests, nonmem_requests

Examples

```
q <- function(value, unit) list(value = value, unit = unit)
model <- compile_model(paste(
  "model one_compartment", "time_unit = h", "parameters",
  "  cl = 0.6 [L/h] log", "  v = 4.5 [L] log", "states", "  central = 0 [mg]",
  "dynamics", "  d/dt(central) = input_central - cl / v * central",
  "outputs", "  cp = central / v [mg/L]", "dosing", "  central [mg] = 1",
  "end", sep = "\n"))
run_template <- list(schema = "qspquest.run/v0.1", solver = "diffsol_bdf",
  budgets = list(solver_callbacks = 1000000L, events = 100L,
                 output_values = 10000L),
  regimen = list(end = q(4, "h"), samples = list(), administrations = list(),
                 rtol = 1e-10, atol = 1e-12))
rows <- do.call(rbind, lapply(c("A", "B"), function(id) data.frame(
  ID = id, TIME = c(0, 1, 2, 4), EVID = c(1L, 0L, 0L, 0L),
  CMT = c(1L, 2L, 2L, 2L), AMT = c(6, 0, 0, 0), MDV = c(1L, 0L, 0L, 0L),
  DV = c(NA, 6 / 4.5 * exp(-0.6 * c(1, 2, 4) / 4.5)))))
dataset <- nonmem_requests(rows, run_template, c("1" = "central"), "h", "mg")
template <- list(schema = "qspquest.fit/v0.1", problem = list(
  kind = "individual", request = list(
    objective = list(simulation = list(schema = "qspquest.sensitivity/v0.1",
        with_respect_to = list("cl"),
        derivative_absolute_tolerances = list(cl = list(central = q(1e-12, "mg.h/L")))),
      observations = list(), priors = list()),
    bounds = list(list(parameter = "cl", lower = q(0.05, "L/h"),
                       upper = q(1.2, "L/h"))),
    max_evaluations = 400L, max_iterations = 100L, gradient_tolerance = 1e-5)))
endpoints <- list("2" = list(output = "cp", value_unit = "mg/L",
  error = list(additive_sd = q(0.1, "mg/L"), proportional_sd = 0)))
prepared <- fit_data_requests(dataset, template, endpoints, mode = "pooled")
fitted <- fit_data(model, prepared)
stopifnot(fitted$results[[1]]$fit$status == "converged")
print(fit_parameter_data_frame(fitted))
print(fit_observation_data_frame(model, fitted))
```

`model_library` *Open the Shared QSPQuest Model Library*

Description

Browse the catalog and execute named scenarios with the model library's verified loader. The library and its Python loader are distributed separately from this package; the QSPQuest Docker Compose deployment provides both.

Usage

```
model_library(root = Sys.getenv("QSPQUEST_LIBRARY_DIR"),
              python = Sys.getenv("QSPQUEST_PYTHON", "python3"))
```

Arguments

root	Directory containing catalog.json, models/ and the qspquest_models loader.
python	Python executable with QSPQuest installed, or access to the QSPQuest CLI.

Details

Pass a nonempty exact version string to pin the catalog selection. The default NULL selects the current catalog version. An unavailable pin fails before simulation and is never replaced automatically. Availability does not establish numerical qualification. The optional shared-library workflow requires Python and the separately distributed model-library package. Native R simulation and fitting do not require Python. Read each model's card and qualification limits before interpreting results. Inspection-only models have no executable scenarios.

Value

A list of functions: models() returns catalog rows; scenarios(model, version = NULL) returns available scenario IDs; run(model, scenario, version = NULL) returns a parsed simulation result, including units, event sides and archive checksum.

Examples

```
# Requires a separately installed QSPQuest model library and Python loader.
if (nzchar(Sys.getenv("QSPQUEST_LIBRARY_DIR"))) {
  library_api <- model_library()
  catalog <- library_api$models()
  length(catalog)
}
```

---

nonmem_requests *Prepare and Simulate Explicit NONMEM-Style Records*

Description

Convert a supported subset of observation and dosing records into one simulation request per subject, preserving original source-row mappings.

Usage

```
nonmem_requests(data, template, compartments, time_unit, amount_unit,
                rate_unit = NULL, observation_side = "post")
simulate_data(model, dataset)
```

Arguments

`data`	A nonempty data frame with ID, TIME and EVID. Optional columns are DV, MDV, AMT, CMT, RATE, ADDL, II and SS. Additional columns are rejected.
`template`	A versioned run request as a list or JSON string. Its regimen administrations, resets, covariate changes, checkpoints, samples and observations must be empty. It supplies start/end, parameters, static covariates, solver, tolerances and per-subject budgets.
`compartments`	Named character vector mapping dose CMT values to model targets.
`time_unit, amount_unit`	Explicit unit strings for TIME/II and AMT.
`rate_unit`	Explicit RATE unit, required for positive RATE.
`observation_side`	Either `"pre"` or `"post"`, applied to every requested observation. Same-time source-row order does not choose event sides.
`model`	A compiled model from `compile_model`.
`dataset`	A `qspquest_dataset` from `nonmem_requests`.

Details

EVID 0 with MDV 0 requests an observation; EVID 1 doses require positive AMT and a mapped CMT. RATE 0 is bolus and positive RATE is finite infusion. ADDL/II specify repeat doses. Missing dose fields and MDV default to zero. DV is retained without inferring missingness: set MDV explicitly. Each subject must have an observation. Duplicate and unsorted observations retain their order. Negative RATE, nonzero SS, EVID 2/3/4 and dose fields on observations are rejected. The engine checks unit compatibility, model targets and the template horizon; conversion never extends the horizon. Subjects execute sequentially. This is not a general NONMEM importer; covariates and censoring are not inferred.

Value

`nonmem_requests` returns a `qspquest_dataset` with per-subject requests, source rows, observation rows and original data. `simulate_data` returns `subjects` with native results and long data, combined `data`, and original `source`. Output includes character ID and one-based SOURCE_ROW. Execution identities remain on each subject result.

See Also

`fit_data_requests`, `result_data_frame`

Examples

```
request <- list(schema = "qspquest.run/v0.1", solver = "diffsol_bdf",
  budgets = list(solver_callbacks = 100000, events = 100, output_values = 1000),
  regimen = list(end = list(value = 4, unit = "h"), samples = list(),
    administrations = list(), rtol = 1e-8, atol = 1e-10))
data <- data.frame(ID = c("A", "A"), TIME = c(0, 1),
  EVID = c(1, 0), AMT = c(6, 0), CMT = c(1, 0))
```

```
prepared <- nonmem_requests(data, request, c("1" = "central"), "h", "mg")
prepared$subjects[[1]]$observation_rows
```

`population_fit_data_requests` *Prepare and Fit Population Data from NONMEM-Style Rows*

Description

Build a FOCEI or SAEM request from an explicit subset of NONMEM records, preserving subject and source-row provenance. Execute the request and extract subject estimates or row-aligned predictions.

Usage

```
population_fit_data_requests(data, template, compartments, endpoints,
                             time_unit, amount_unit, rate_unit = NULL,
                             observation_side = "post")
fit_population_data(model, dataset)
population_fit_subject_data_frame(fitted)
population_fit_observation_data_frame(fitted)
```

Arguments

`data`	A nonempty data frame with unique columns and required ID, TIME and EVID. Supported optional columns are DV, MDV, AMT, CMT, RATE, ADDL, II and SS.
`template`	A versioned `qspquest.fit/v0.1` FOCEI or SAEM request, as a list or JSON string. It must contain exactly one subject objective with empty observations and priors, a sensitivity selection and a run whose regimen event lists are empty. The subject serves as the common simulation template.
`compartments`	Named character vector mapping dose CMT values to model targets.
`endpoints`	Named list mapping observation CMT values to lists containing exactly `output`, `value_unit` and `error`. Each error contains an explicit additive_sd quantity and finite proportional_sd.
`time_unit, amount_unit, rate_unit`	Explicit unit strings. A positive RATE requires rate_unit.
`observation_side`	Either `"pre"` or `"post"`. Source-row order does not decide dose-side semantics.
`model`	A model from `compile_model`.
`dataset`	The prepared population dataset returned by `population_fit_data_requests`.
`fitted`	A result returned by `fit_population_data`.

Details

Only EVID 0 observations and EVID 1 doses are supported; MDV 1 omits an observation. Included DV values must be finite. Nonzero SS, negative RATE, unmapped compartments/endpoints, extra columns, covariate records and censoring are rejected. The native engine performs model, unit, time and fitting checks. FOCEI and SAEM objectives differ and their numerical values are not directly interchangeable.

Value

`population_fit_data_requests` returns a prepared request, subject IDs, the original data frame and a mapping. SOURCE_ROW is one-based in R; SUBJECT_INDEX, OBSERVATION_INDEX and RESULT_ROW are zero-based native indices. `fit_population_data` returns the native versioned result and that mapping, after checking subject and fitted-observation alignment. `population_fit_subject_data_fr` returns one row per subject and random effect with ID, eta, fixed effect, derived individual estimate, parameter unit, conditional negative log likelihood and fit status. `population_fit_observation_data_frame` returns FOCEI fitted observations with ID, source and native row indices, time, side, DV, PRED, IPRED, units and residuals. It returns `NULL` when SAEM has no fitted observations. Residuals are missing when the compared units differ.

See Also

`nonmem_requests`, `fit_data_requests`

Examples

```
q <- function(value, unit) list(value = value, unit = unit)
model <- compile_model(paste(
  "model one_compartment", "time_unit = h", "parameters",
  "  cl = 0.6 [L/h] log", "  v = 4.5 [L] log", "states", "  central = 0 [mg]",
  "dynamics", "  d/dt(central) = input_central - cl / v * central",
  "outputs", "  cp = central / v [mg/L]", "dosing", "  central [mg] = 1",
  "end", sep = "\n"))
run_template <- list(schema = "qspquest.run/v0.1", solver = "diffsol_bdf",
  budgets = list(solver_callbacks = 1000000L, events = 100L,
                 output_values = 10000L),
  regimen = list(end = q(4, "h"), samples = list(), administrations = list(),
                 rtol = 1e-10, atol = 1e-12))
etas <- c(-0.22, -0.08, 0.08, 0.22)
rows <- do.call(rbind, lapply(seq_along(etas), function(i) data.frame(
  ID = paste0("S", i), TIME = c(0, 1, 2, 4), EVID = c(1L, 0L, 0L, 0L),
  CMT = c(1L, 2L, 2L, 2L), AMT = c(6, 0, 0, 0), MDV = c(1L, 0L, 0L, 0L),
  DV = c(NA, 6 / 4.5 * exp(-0.6 * exp(etas[i]) * c(1, 2, 4) / 4.5)))))
simulation <- list(schema = "qspquest.sensitivity/v0.1", run = run_template,
  with_respect_to = list("cl"),
  derivative_absolute_tolerances = list(cl = list(central = q(1e-12, "mg.h/L"))))
simulation$run$parameters <- list(cl = q(0.5, "L/h"), v = q(4.5, "L"))
template <- list(schema = "qspquest.fit/v0.1", problem = list(
  kind = "focei", request = list(
    subjects = list(list(simulation = simulation, observations = list(),
                         priors = list())),
    fixed_effects = list(list(parameter = "cl", lower = q(0.35, "L/h"),
```

```
                                upper = q(0.85, "L/h"))),
      random_effects = list(list(parameter = "cl")),
      omega = list(list(0.04)), omega_diagonal_bounds = list(list(0.04, 0.04)),
      eta_bound = 1.5, max_evaluations = 30000L, max_iterations = 80L,
      gradient_tolerance = 0.005, seed = 17L)))
  endpoints <- list("2" = list(output = "cp", value_unit = "mg/L",
    error = list(additive_sd = q(0.05, "mg/L"), proportional_sd = 0)))
  prepared <- population_fit_data_requests(rows, template, c("1" = "central"),
    endpoints, time_unit = "h", amount_unit = "mg")
  fitted <- fit_population_data(model, prepared)
  stopifnot(fitted$result$population$status == "converged")
  print(population_fit_subject_data_frame(fitted))
  print(head(population_fit_observation_data_frame(fitted)))
```

`qspquest_export_rxode2_ui` *Export an rxode2 UI for Structural QSPQuest Import*

Description

Exports model syntax, numeric fixed-effect initial values, state order and runtime versions. Install rxode2 separately. The snapshot is read by the `import_rxode2_snapshot` adapter of the separately distributed QSPQuest Python package, which checks its own supported structural subset; this helper does not run a fit or qualify population-estimation equivalence.

Usage

```
qspquest_export_rxode2_ui(ui, path)
```

Arguments

`ui`	An existing rxode2/nlmixr2-compatible UI object with an explicit model block.
`path`	Destination JSON file.

Value

The snapshot invisibly, after writing its JSON representation.

Examples

```
if (requireNamespace("rxode2", quietly = TRUE)) {
  ui <- rxode2::rxode2(function() {
    ini({ ka <- 1.1; cl <- 0.3; v <- 3 })
    model({
      d/dt(depot) <- -ka * depot
      d/dt(central) <- ka * depot - cl / v * central
      cp <- central / v
    })
  })
```

```
  path <- tempfile(fileext = ".json")
  snapshot <- qspquest_export_rxode2_ui(ui, path)
  names(snapshot$parameters)
  unlink(path)
}
```

`result_data_frame` *Convert a Simulation Result to Long Data*

Description

Produce long-format output suitable for plotting or joining with observation records. Missing values remain present as `NA`.

Usage

```
result_data_frame(result)
```

Arguments

`result`	A parsed `qspquest.result/v0.1` simulation result.

Value

A data frame with `TIME`, `SIDE`, `OUTPUT`, `VALUE` and `UNIT` columns. Attributes `time_unit` and `execution_identity` retain simulation metadata. Rows are grouped by output, retaining the native observation order within each output.

See Also

`simulate_model`, `simulate_data`

Examples

```
# A parsed simulation result, as returned by simulate_model().
result <- list(schema = "qspquest.result/v0.1", time_unit = "h",
  times = list(0, 1), sides = list("post", "post"),
  outputs = list(list(name = "cp", unit = "mg/L", values = list(2, 1.81))),
  identity = list())
result_data_frame(result)
```

Index

compile_model, 3, *3*

fit_data *(*fit_data_requests*)*, 4
fit_data_requests, *3*, 4, *8*, *10*, *12*
fit_gamma_poisson, *3*, 6
fit_model, *5*
fit_model *(*compile_model*)*, 3
fit_observation_data_frame
        *(*fit_parameter_data_frame*)*, 7
fit_parameter_data_frame, *5*, 7
fit_population_data
        *(*population_fit_data_requests*)*,
        11
fit_probit_normal_binomial
        *(*fit_gamma_poisson*)*, 6

model_library, 8
morris_model *(*compile_model*)*, 3

nonmem_requests, *3*, *5*, *8*, 9, *12*

population_fit_data_requests, *3*, 11
population_fit_observation_data_frame
        *(*population_fit_data_requests*)*,
        11
population_fit_subject_data_frame
        *(*population_fit_data_requests*)*,
        11

qspquest *(*qspquest-package*)*, 2
qspquest-package, 2
qspquest_export_rxode2_ui, 13

result_data_frame, *3*, *10*, 14

scan_model *(*compile_model*)*, 3
simulate_data, *14*
simulate_data *(*nonmem_requests*)*, 9
simulate_model, *14*
simulate_model *(*compile_model*)*, 3