Package 'nlmixr2utils'

September 9, 2026

Title Shared Infrastructure for 'nlmixr2' Extension Packages

Version 0.3.1

Description Provides shared worker-plan helpers, covariance utilities, model reexports, equation rendering methods, and package data used by the split 'nlmixr2' extension packages, including 'nlmixr2boot', 'nlmixr2llp', 'nlmixr2scm', and 'nlmixr2sir'. These helpers give the extension packages a common canonical raw-results schema, run-cache and seeding infrastructure, and equation-printing methods so that each extension package does not need to reimplement this shared functionality separately.

Depends R (>= 4.0)

License GPL (>= 3)

URL https://github.com/nlmixr2/nlmixr2utils

BugReports https://github.com/nlmixr2/nlmixr2utils/issues

Imports cli (>= 3.4.0), future, future.apply, jsonlite, knitr, methods, nlmixr2est (>= 5.0.0), parallel, rxode2 (>= 5.0.0)

Suggests nlmixr2data, progressr, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Justin Wilkins [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-7099-9396>), Matthew Fidler [aut] (ORCID: <https://orcid.org/0000-0001-8538-6691>), Vipul Mann [aut], Vishal Sarsani [aut] (ORCID: <https://orcid.org/0000-0002-9272-3438>), Christian Bartels [ctb], Bill Denney [aut] (ORCID: <https://orcid.org/0000-0002-5759-428X>)

Maintainer Justin Wilkins <justin.wilkins@occams.com>

Repository CRAN

Date/Publication 2026-09-09 12:30:02 UTC

Contents

`.plap` *Apply FUN over X with optional parallel execution and progress reporting*

Description

When **future.apply** is available, uses `future_lapply()` under the current `future::plan()`. If **progressr** is also available, progress is reported. Otherwise falls back to `base::lapply()`.

Usage

```
.plap(X, FUN, ..., rxThreads = NULL, .label = NULL)
```

Arguments

`X`	vector or list to iterate over
`FUN`	function applied to each element of `X`
`...`	additional arguments passed to `FUN`
`rxThreads`	optional; when not `NULL`, every call to `FUN` captures its own worker's current `rxode2::getRxThreads()` value, sets `rxode2::setRxThreads(rxThreads)`, runs `FUN`, and restores the captured value on exit – applied identically whether the call runs in the main process or inside a parallel worker, and safe for persistent `multisession` workers reused across separate `.plap()` calls (a later call does not inherit an earlier call's setting).
`.label`	optional `function(x) -> character` producing a per-item progress label; `x` is each element of `X`

Value

list of results in the same order as `X`

Examples

```
.plap(1:3, function(x) x * 2)
```

`.resolveEffectiveWorkers` *Resolve the effective number of parallel workers for a workers= specification, without changing the current future plan*

Description

Resolve the effective number of parallel workers for a workers= specification, without changing the current future plan

Usage

```
.resolveEffectiveWorkers(workers)
```

Arguments

`workers`	`NULL`, `"auto"`, or a positive integer – same values accepted by `.withWorkerPlan()`'s `workers` argument.

Value

integer; the number of workers that would actually be used. For `NULL`, this reflects the number of workers in the *currently active* `future` plan (`1L` for a `sequential` plan), not a hypothetical future one.

Examples

```
.resolveEffectiveWorkers(4L)
.resolveEffectiveWorkers(NULL)
```

`.validateRxThreads` *Validate an rxode2-threads-per-worker specification*

Description

Validate an rxode2-threads-per-worker specification

Usage

```
.validateRxThreads(rxThreads)
```

Arguments

rxThreads	NULL, "auto", 1, or a positive integer.

Value

Invisibly returns NULL when rxThreads is valid.

Examples

```
.validateRxThreads(1L)
.validateRxThreads("auto")
```

.validateWorkers *Validate a worker-plan specification*

Description

Validate a worker-plan specification

Usage

```
.validateWorkers(workers)
```

Arguments

workers	NULL, "auto", 1, or a positive integer.

Value

Invisibly returns NULL when workers is valid.

Examples

```
.validateWorkers(1L)
.validateWorkers("auto")
```

`.withWorkerPlan` *Temporarily set a future parallel plan for the duration of an expression*

Description

Also guards against requesting more OS threads than the machine has, whenever more than one worker is involved: the *effective* number of workers (from `workers`, or from the ambient `future` plan when `workers = NULL`) times the *effective* number of rxode2 threads per worker (from `rxThreads`, or from `rxode2::getRxThreads()` when `rxThreads = NULL`) must not exceed the total core count, or the call aborts before anything is evaluated or any global state is changed. A single worker (the default, sequential case) is never subject to this check – it cannot oversubscribe by definition.

Usage

```
.withWorkerPlan(workers, expr, rxThreads = NULL)
```

Arguments

`workers`	`NULL` (leave the current plan unchanged), `1` (force sequential), a positive integer (use that many `multisession` workers), or `"auto"` (use `future::availableCores(omit = 1)`).
`expr`	expression to evaluate; the prior plan is always restored on exit, even if `expr` throws an error. The main session's rxode2 thread count is restored the same way; thread counts broadcast into an *ambient* plan's existing workers (i.e. when `workers = NULL`) are not reverted, since this function never created or owns that plan.
`rxThreads`	`NULL` (use the current `rxode2::getRxThreads()` value), `"auto"` (divide the total core count evenly across the effective worker count), or a positive integer – the rxode2 thread count applied in the main session for the duration of `expr`, and best-effort broadcast into every worker of the active plan (whether newly created by this call or already ambient). `.plap()`'s own `rxThreads` argument remains the authoritative per-task mechanism for callers that use it; this broadcast exists so callers that do not still get an accurate thread count applied.

Value

value of `expr`

Examples

```
.withWorkerPlan(NULL, 1 + 1)
```

`knit_print.nlmixr2FitCore` *Extract the equations from an nlmixr2/rxode2 model to produce a 'LaTeX' equation.*

Description

Extract the equations from an nlmixr2/rxode2 model to produce a 'LaTeX' equation.

Usage

```
## S3 method for class 'nlmixr2FitCore'
knit_print(x, ..., output = "equations")

## S3 method for class 'rxUi'
knit_print(x, ...)
```

Arguments

`x`	The model to extract equations from
`...`	Ignored
`output`	The type of output to request (currently, just "equations")

Value

A `knitr::asis_output()` object containing the 'LaTeX' equation block, ready to be printed as-is inside a 'knitr'/'rmarkdown' document.

Examples

```
mod <- function() {
  ini({
    lka <- 0.45
    lcl <- 1
    lvc <- 3.45
    propSd <- 0.5
  })
  model({
    ka <- exp(lka)
    cl <- exp(lcl)
    vc <- exp(lvc)
    cp <- linCmt()
    cp ~ prop(propSd)
  })
}
ui <- rxode2::rxode(mod)
knit_print(ui)
```

raw-results *Canonical raw-results helpers*

Description

These helpers define the shared per-fit raw-results schema used by the nlmixr2 extension packages. Writers emit the canonical column order, optional standard-error columns, and a JSON sidecar describing the block boundaries so downstream readers do not need to re-parse parameter labels.

Usage

```
rawResultsSchema(fit)

rawResultsRow(
  fit = NULL,
  source,
  hypothesis,
  sample,
  modelLabel,
  role,
  errorMessage = NA_character_,
  objf = NULL,
  minimizationSuccessful = NULL,
  covarianceStepSuccessful = NULL,
  estimateNearBoundary = NULL,
  significantDigits = NULL,
  conditionNumber = NULL,
  theta = NULL,
  omega = NULL,
  sigma = NULL,
  se = NULL,
  schema = NULL
)

writeRawResults(rows, dir, basename = "raw_results")

readRawResults(path)

setupRawResultsFilter(filter)

parseRawResultsParams(rawres, fit, offset = 1L, filter = NULL)
```

Arguments

fit	A fitted nlmixr2 object, or a fit-like list containing theta, omega, sigma, and related metadata.
source	Canonical producer name such as "bootstrap", "sir", or "sse".

`hypothesis` Hypothesis label recorded in the raw-results row.

`sample` Integer replicate index. Use `0` for a reference row.

`modelLabel` Short model label stored in `model_label`.

`role` Role label stored in `role`.

`errorMessage` Optional error message recorded for failed fits.

`objf` Optional objective-function value override.

`minimizationSuccessful`, `covarianceStepSuccessful`, `estimateNearBoundary`
Optional diagnostic flag overrides. When `NULL`, `rawResultsRow()` uses simple heuristics based on the supplied fit.

`significantDigits`
Optional significant-digits override.

`conditionNumber`
Optional condition-number override.

`theta`, `omega`, `sigma`
Optional parameter overrides. `theta` may be a named vector or an unnamed vector in schema order. `omega` and `sigma` may be named vectors, unnamed vectors in schema order, or full matrices.

`se` Optional standard-error overrides in parameter-column order.

`schema` Optional schema list from `rawResultsSchema()`. Supply this to target a wider union schema than the one implied by `fit`.

`rows` Data frame of canonical raw-results rows.

`dir` Directory where the CSV, RDS, and JSON sidecar should be written.

`basename` Basename used for the output files.

`path` Path to a canonical raw-results CSV, RDS, or containing directory.

`filter` Either a PsN-style character filter, a one-sided formula, or an unevaluated expression.

`rawres` A raw-results data frame, a path understood by `readRawResults()`, or the result of `readRawResults()`.

`offset` Integer sample offset. The default `1L` skips the canonical reference row with `sample = 0`.

Value

`rawResultsSchema()` returns a list with `columns`, `baseCols`, `thetaCols`, `omegaCols`, `sigmaCols`, `seCols`, and `schemaVersion`.

`rawResultsRow()` returns a one-row data frame in canonical schema order.

`writeRawResults()` invisibly returns a list containing the canonical data frame and the three output paths.

`readRawResults()` returns the raw-results data frame with the parsed header attached as the `rawResultsHeader` attribute.

`setupRawResultsFilter()` returns a predicate function that accepts a raw-results data frame and returns a logical inclusion vector.

`parseRawResultsParams()` returns a named list of per-sample parameter sets, each containing `sample`, `source`, `hypothesis`, `modelLabel`, `role`, `theta`, `omega`, and `sigma`.

Lifecycle

Stable.

`resolveRxThreads` *Resolve the effective rxode2 threads-per-worker for an rxThreads= specification*

Description

Validates **both** arguments – `workers` first, then `rxThreads` – so an invalid `workers` value is rejected cleanly here rather than reaching `.resolveEffectiveWorkers()`'s `as.integer()` coercion with garbage input.

Usage

```
resolveRxThreads(workers, rxThreads = NULL)
```

Arguments

`workers`	the same `workers` value passed to the caller's own `workers=` argument – used only to compute `rxThreads = "auto"`. Note that `workers` is validated on **every** call, even when the branch taken doesn't otherwise use it (`rxThreads = NULL` or an explicit integer) – so a caller passing an invalid `workers` alongside a perfectly valid `rxThreads` still gets an error about `workers`. This is intentional defense-in-depth.
`rxThreads`	`NULL` (use the current `rxode2::getRxThreads()` value), `"auto"` (divide the total core count evenly across `workers`, minimum 1), or a positive integer.

Value

integer; the rxode2 thread count that would actually be used.

Examples

```
resolveRxThreads(NULL, NULL)
resolveRxThreads(4L, "auto")
```

run-cache *Shared run-cache helpers*

Description

These helpers implement the common numbered-run-directory, state-file, task-cache, and deterministic seeding patterns used by the split nlmixr2 extension packages.

Usage

```
resolveRunDir(prefix, fitName, restart, outputDir = NULL)

writeRunState(dir, state, schema)

readRunState(dir, schema)

taskCache(dir, key = NULL)

pendingTasks(allKeys, completedKeys)

withRunSeed(dir, seed = NULL, key = NULL, prefix = NULL, expr)

deriveFitName(expr, maxLength = 50L)
```

Arguments

prefix	Optional seed-file prefix override. Supply this when you need a stable artifact name such as "boot" or "sir" regardless of the directory basename.
fitName	Base fit name used when constructing numbered output directories.
restart	Logical flag controlling resume versus fresh-run behavior. With outputDir = NULL, restart = FALSE resumes the latest numbered directory when one exists; otherwise a new numbered directory is selected. With an explicit outputDir, restart = TRUE marks an existing directory for overwrite.
outputDir	Optional explicit output directory.
dir	Run directory containing the state file.
state	Arbitrary R object to persist.
schema	Either a prefix string, for example "sse", or a list with prefix and optional version.
key	Optional cache namespace used to isolate one family of artifacts inside a run directory.
allKeys	Full set of task keys.
completedKeys	Completed task keys.
seed	Optional explicit master seed. When NULL, the helper derives a stable integer seed from the current .Random.seed if one exists, and persists it for later resumes.

expr	Optional expression to evaluate under a derived per-key seed.
maxLength	Maximum length of the returned fit name.

Value

resolveRunDir() returns a list with path, mode, created, and prefix.

writeRunState() invisibly returns the state-file path; readRunState() returns the saved state or NULL when the file does not exist.

taskCache() returns a list with cache methods has(), get(), put(), and keys().

pendingTasks() returns the ordered subset of allKeys that does not appear in completedKeys.

With no key, withRunSeed() returns the persisted master seed. With key and no expr, it returns the derived per-key seed. With both key and expr, it evaluates expr under that derived seed and restores the prior RNG state on exit.

deriveFitName() returns a sanitized single-string fit label.

Lifecycle

Stable.

---

setQuietFastControl *Make an estimation control object quieter and faster*

Description

setQuietFastControl() is a small utility for repeated model evaluations where full iteration printing, covariance estimation, table generation, and object compression are unnecessary. It returns the input control object after forcing a small set of fields to faster, quieter defaults.

Usage

```
setQuietFastControl(ctl)
```

Arguments

ctl	A control object, typically a named list passed to an nlmixr2() estimation routine.

Details

Specifically, it sets:

- print = 0
- covMethod = 0
- calcTables = FALSE
- compress = FALSE

Value

The modified control object.

Examples

```
ctl <- list(
  print = 100,
  covMethod = "r,s",
  calcTables = TRUE,
  compress = TRUE
)

setQuietFastControl(ctl)
```

`theoFitOde` *Example single dose Theophylline ODE model*

Description

This is a nlmixr2 model that is pre-run so that it can be used in package testing and development. It is regenerated when `devtools::document()` is run on a source checkout. If there is a binary incompatability between the fit objects, re-documenting (or running `build/build.R`) will fix this nlmixr2 fit object.

Format

A (modified) data frame with 132 rows and 22 columns.

ID Patient identifier

TIME Time (hr)

DV Dependent variable (concentration)

PRED Predictions without any between subject variability

RES Population Residual

WRES Weighted Residuals under the FO assumption

IPRED Individual Predictions

IRES Individual Residuals

IWRES Individual Weighted Residuals

CPRED Conditional Prediction under the FOCE assumption

CRES Conditional Residuals under the FOCE assumption

CWRES Conditional Weighted Residuals under the FOCE assumption

eta.ka Between subject changes for ka

eta.cl Between subject changes for cl

eta.v Between subject changes for v

depot amount in the depot compartment

center amount in the central compartment

ka Individual ka values

cl Individual cl values

v Individual volume of distribution

tad Time after dose

dosenum Dose number

Index