

Package: nlmixr2targets (via r-universe)

October 20, 2026

Title Targets for 'nlmixr2' Pipelines

Version 0.1.0.9000

Description 'nlmixr2' often has long runtimes. A pipeline toolkit tailored to 'nlmixr2' workflows leverages 'targets' and 'nlmixr2' to ease reproducible workflows. 'nlmixr2targets' ensures minimal rework in model development with 'nlmixr2' and 'targets' by simplifying and standardizing models and datasets.

License GPL (>= 2)

URL <https://nlmixr2.github.io/nlmixr2targets/>

BugReports <https://github.com/nlmixr2/nlmixr2targets/issues>

Depends R (>= 4.1)

Imports checkmate, digest, nlmixr2est, rxode2 (>= 2.0.14), targets

Suggests covr, knitr, nlmixr2data, rmarkdown, spelling, tarchetypes, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/pak/sysreqs cmake libglib-dev libgmp3-dev make libicu-dev libxml2-dev

Repository <https://nlmixr2.r-universe.dev>

Date/Publication 2026-10-18 20:28:15 UTC

RemoteUrl <https://github.com/nlmixr2/nlmixr2targets>

RemoteRef HEAD

RemoteSha 9dd57d714d13410ab4f023c3f5782bf9da15ae66

Contents

assign_origData	2
nlmixr_data_simplify	3
nlmixr_object_complicate	4
nlmixr_object_covariates	5
nlmixr_object_simplify	6
nlmixr_object_zero_initial_eval	7
nlmixr2_indirect	8
nlmixr2targets_cache_prune	10
nlmixr2targets_cache_status	11
print.nlmixr2targetsError	11
tar_nlmixr	12
tar_nlmixr_multimodel	19
tar_nlmixr_multimodel_has_self_reference	23
tar_nlmixr_multimodel_parse	24
tar_nlmixr_multimodel_prep	26
tar_nlmixr_multimodel_single	27
Index	30

assign_origData	<i>Replace the fit data with the original data, then return the modified fit</i>
-----------------	--

Description

This function is intended for use within nlmixr2targets target creation, and it’s not typically invoked by users.

Usage

```
assign_origData(fit, data)
```

Arguments

- fit an estimated nlmixr2 object
- data the data from the original fit

Value

The fit with the data added back in as fit\$env\$origData

nlmixr_data_simplify *Standardize and simplify data for nlmixr2 estimation*

Description

This function is typically not needed by end users.

Usage

```
nlmixr_data_simplify(
  data,
  object,
  table = list(),
  directory = file.path(targets::tar_config_get("store"), "user/nlmixr2"),
  est = NULL,
  control = NULL,
  covariates = NULL
)
```

Arguments

data	nlmixr data
object	The model, in either form <code>nlmixr_object_covariates()</code> accepts: an nlmixr ui object, or a character md5 hash identifying a simplified ui in the <code>nlmixr2targets</code> indirect cache.
table	The output table control object (like <code>'tableControl()'</code>)
directory	Cache directory to load the simplified nlmixrui from. Defaults to <code>file.path(targets::tar_config_get("store"), "user/nlmixr2")</code> , mirroring the convention used by targets' own <code>store = targets::tar_config_get("store")</code> defaults — the path resolves at call time against whatever store the user has configured for the running <code>tar_make()</code> (e.g. a <code>tempdir()</code> location set via <code>targets::tar_config_set()</code>).
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
covariates	The model's covariate column names, as returned by <code>nlmixr_object_covariates()</code> . Supply exactly one of object and covariates.

Details

The standardization keeps columns that rxode2 and nlmixr2 use along with the covariates. Column order is standardized (rxode2 then nlmixr2 then alphabetically sorted covariates), and rxode2 and nlmixr2 column names are converted to lower case.

`est` and `control` only affect which columns are kept: with the default `est = NULL`, or any estimation method other than "vae", the standard and model covariate columns above are all that is kept. When

`est = "vae"`, the automated covariate selection searches subject-constant data columns beyond the covariates named in the model, so the candidate columns reported by `nlmixr2est::vaeCovariates()` (honoring the `shapes`, `covCenterType`, `covCenter`, and `catCutoff` settings in `control`) are kept as well. Columns the search cannot use (for example time-varying or partially-missing columns) are still dropped; the exclusion warnings that `nlmixr2est::nlmixr()` would raise for them are raised here instead, because the estimation step never sees the dropped columns.

The model is consulted only for its covariate names, so it can be given either as object or, already reduced to those names, as `covariates`. The targets generated by `tar_nlmixr()` pass `covariates`, so the data target depends on the covariate names rather than on the whole model: an edit to the model that leaves its covariates unchanged does not rebuild the simplified data.

Value

The data with the `nlmixr2` column lower case and on the left and the covariate columns on the right and alphabetically sorted.

See Also

Other Simplifiers: `nlmixr_object_complicate()`, `nlmixr_object_covariates()`, `nlmixr_object_simplify()`

`nlmixr_object_complicate`

Re-attach labels, metadata, and original data to a simplified fit

Description

The inverse of `nlmixr_object_simplify()`. Given a fit produced from the simplified, label-and-meta-stripped model, plus the original model function and the original data, this function:

Usage

```
nlmixr_object_complicate(fit, object, data)
```

Arguments

<code>fit</code>	An estimated <code>nlmixr2</code> fit produced from the simplified model.
<code>object</code>	The original model function (or object) the fit was derived from. Its labels and metadata are read back onto <code>fit</code> .
<code>data</code>	The original data the fit corresponds to (before <code>nlmixr_data_simplify()</code> reduced its column set). Must have the same number of rows as the data currently stored on the fit, mirroring <code>assign_origData()</code> .

Details

- re-derives parameter labels and the metadata environment from object and writes them back onto `fit$ui$iniDf$label` and `fit$ui$meta`,
- rebuilds the Parameter (label) column of the cached parameter tables (`fit$env$parFixed` and `fit$env$parFixedDf`). `nlmixr2est` computes those tables once, at estimation time – when the labels were still stripped – so restoring `iniDf$label` alone would not make the labels show when the fit is printed, and
- replaces `fit$env$origData` with the original data.

This is what makes label/meta edits in the source model cheap under targets: the cache hash for the simplified model object is independent of labels and metadata, so `tar_make()` only re-runs this re-attachment step (and the `cheap_object_simple` step) when only labels or metadata change.

This function is typically not invoked directly by end users; it is the command for the final target produced by `tar_nlmixr()`.

Value

The modified fit.

See Also

`nlmixr_object_simplify()`, `assign_origData()`.

Other Simplifiers: `nlmixr_data_simplify()`, `nlmixr_object_covariates()`, `nlmixr_object_simplify()`

`nlmixr_object_covariates`

The covariate columns a simplified model reads from the data

Description

This function is typically not needed by end users. It is the command of the `_covariates` target generated by `tar_nlmixr()`, whose value is what `nlmixr_data_simplify()` keeps beyond the standard event columns.

Usage

```
nlmixr_object_covariates(
  object,
  directory = file.path(targets::tar_config_get("store"), "user/nlmixr2")
)
```

Arguments

object	Either an nlmixr ui object (e.g. the output of running <code>nlmixr(object = model)</code>) or a character md5 hash identifying a simplified ui in the <code>nlmixr2targets</code> indirect cache (the form used in targets generated by <code>tar_nlmixr()</code>).
directory	Cache directory to load the simplified nlmixrui from. Defaults to <code>file.path(targets::tar_config_get("user/nlmixr2"))</code> , mirroring the convention used by targets' own <code>store = targets::tar_config_get("store")</code> defaults — the path resolves at call time against whatever store the user has configured for the running <code>tar_make()</code> (e.g. a <code>tempdir()</code> location set via <code>targets::tar_config_set()</code>).

Details

The names are de-duplicated and sorted in the C locale, so reordering the lines of a model, or running on a worker with a different locale, does not change the value and does not rebuild the simplified data.

Value

A sorted character vector of covariate names, possibly empty.

See Also

Other Simplifiers: `nlmixr_data_simplify()`, `nlmixr_object_complicate()`, `nlmixr_object_simplify()`

```
nlmixr_object_simplify
```

Simplify an nlmixr object

Description

This function is typically not needed by end users.

Usage

```
nlmixr_object_simplify(
  object,
  directory = file.path(targets::tar_config_get("store"), "user/nlmixr2")
)
```

Arguments

object	Fitted object or function specifying the model.
directory	Cache directory to load the simplified nlmixrui from. Defaults to <code>file.path(targets::tar_config_get("user/nlmixr2"))</code> , mirroring the convention used by targets' own <code>store = targets::tar_config_get("store")</code> defaults — the path resolves at call time against whatever store the user has configured for the running <code>tar_make()</code> (e.g. a <code>tempdir()</code> location set via <code>targets::tar_config_set()</code>).

Details

The object simplification removes comments (so please use `label()` instead of comments to label parameters) and then converts the object to a "nlmixrui" object.

Object metadata (`ui$meta`) and parameter labels (`ui$iniDf$label`) are also stripped from the simplified object before it is written to the indirect cache. They do not affect estimation, and stripping them keeps the cache hash stable across edits to either, so editing only labels or metadata will not invalidate the cached fit. The stripped values are restored on the final fit by `nlmixr_object_complicate()`, which reads them straight back off the original model.

The natural nlmixr2 DSL form for compartment initial conditions (`cmt(0) <- value` inside a `model({...})` block) trips targets' static analysis because `codetools::findGlobals()` interprets it as a replacement-function assignment with a non-symbol target. `tar_nlmixr()` auto-rewrites `cmt(0) <- value` to `cmt(initial) <- value` inside `model({...})` blocks at construction time (mutating the user's model function in `env`), and converts back to `cmt(0) <- value` before nlmixr2 sees the model. The user can therefore write `cmt(0) <- value` directly.

Manual `cmt(initial) <- value` is also accepted, but it is a nlmixr2targets-only workaround: bare nlmixr2 does not understand the `cmt(initial)` form, so a model function written this way only fits when routed through `tar_nlmixr()` / `tar_nlmixr_multimodel()`. Note: because the rewrite mutates the function in `env`, calling the model function directly (outside `tar_make()`) after `tar_nlmixr()` will see `cmt(initial)` in its body.

The simplified model's `model.name` is always set to "object". This keeps the simplified output stable so that the MD5 hash used by the targets indirect cache is independent of the symbol the caller bound the model function to.

Value

The MD5 hash used to load the simplified nlmixrui object back from the nlmixr2targets indirect cache.

See Also

`nlmixr_object_complicate()` for the inverse operation that re-attaches labels, metadata, and the original data on the final fit.

Other Simplifiers: `nlmixr_data_simplify()`, `nlmixr_object_complicate()`, `nlmixr_object_covariates()`

nlmixr_object_zero_initial_eval

Runtime helper: undo the `cmt(0) -> cmt(initial)` rewrite before evaluating the captured object expression.

Description

Wrapped around the captured expression at construction time when `tar_nlmixr_protect_zero_initial()` performed any rewrites. Needed for pipe forms like `pheno |> model({...})` and `pheno |> ini(...)`, because nlmixr2's pipe handlers parse `pheno`'s body before the existing `nlmixr_object_simplify_zero_initial_helper()` could intervene. Harmless for symbol-only forms (the existing helper reaches them at simplify time anyway).

Usage

```
nlmixr_object_zero_initial_eval(expr, envir = parent.frame())
```

Arguments

expr	A quoted (unevaluated) language object.
envir	The parent environment for evaluation. Defaults to the caller's frame (the targets execution env).

Details

Mechanism:

- Walk the quoted expression, rewriting `name(initial) <- val` back to `name(0) <- val` via the existing inverse helper.
- Walk the (rewritten) expression for symbols. For each that resolves to a function whose body still contains `name(initial) <- val` inside `model({...})`, build a corrected closure with `name(0) <- val` and bind it in an override frame.
- Evaluate the rewritten expression in the override frame (whose parent is `envir`), so any lookup of a rewritten symbol hits the corrected closure first.

The corrected copies retain the original closure environment of the user's function; only the body is swapped.

Value

The result of evaluating the corrected expression.

nlmixr2_indirect	<i>Estimate an nlmixr2 model loading the model from a targets indirect hash storage</i>
------------------	---

Description

This is the command used by the `fit_simple` target generated by `tar_nlmixr()`; it is not typically invoked by end users.

Usage

```
nlmixr2_indirect(
  object,
  data,
  est,
  control,
  table,
  directory = file.path(targets::tar_config_get("store"), "user/nlmixr2"),
  error = c("stop", "continue"),
  description = NULL
)
```

Arguments

object	Fitted object or function specifying the model.
data	nlmixr data
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
table	The output table control object (like <code>'tableControl()'</code>)
directory	Cache directory to load the simplified nlmixrui from. Defaults to <code>file.path(targets::tar_config_get("user/nlmixr2"))</code> , mirroring the convention used by targets' own <code>store = targets::tar_config_get("store")</code> defaults — the path resolves at call time against whatever store the user has configured for the running <code>tar_make()</code> (e.g. a <code>tempdir()</code> location set via <code>targets::tar_config_set()</code>).
error	What should happen if the estimation step throws an error? <code>"stop"</code> (the default) lets the error propagate, halting <code>targets::tar_make()</code> as usual. <code>"continue"</code> catches the error and stores a failure sentinel (an object of class <code>nlmixr2targetsError</code> , which also inherits from <code>"try-error"</code>) carrying the error message, so a single failed model does not stop the rest of the pipeline. Detect a failed fit with <code>inherits(fit, "nlmixr2targetsError")</code> or the broader <code>inherits(fit, "try-error")</code> .
description	Human-readable name for the model, announced by the estimation target when it starts to run. <code>NULL</code> (the default) announces nothing.

Details

object is the md5 hash returned by `nlmixr_object_simplify()` and stored as the `object_simple` target in the pipeline. `nlmixr2_indirect()` loads the simplified nlmixrui object from the `nlmixr2targets` indirect cache (under `<targets store>/user/nlmixr2/`) and passes it to `nlmixr2est::nlmixr()` along with `data`, `est`, `control`, and – when the caller supplied one – `table`. An omitted `table` is passed on as omitted rather than as `nlmixr2est::tableControl()`, because `nlmixr2est` merges the `table` settings carried on `ui$meta` into the `table` control only when its own `table` argument is missing.

Routing the simplified `ui` through a small character-hash target rather than a target whose value is the `ui` object keeps the dependency hash for the `fit_simple` target independent of large `rxUi` internals, so cosmetic edits to labels or metadata (which `nlmixr_object_simplify()` strips before caching) do not invalidate the cached fit.

Value

An `nlmixr2` fit object, as returned by `nlmixr2est::nlmixr()`. When `error = "continue"` and the estimation step fails, a failure sentinel of class `nlmixr2targetsError` (which also inherits from `"try-error"`) is returned instead; see `tar_nlmixr()`.

See Also

`tar_nlmixr()`, `nlmixr_object_simplify()`.

nlmixr2targets_cache_prune

Drop orphaned entries from the nlmixr2targets indirect cache

Description

Removes cache entries whose md5 hash is not in keep. By default dry_run = TRUE, so the function reports what would be removed without actually deleting anything; pass dry_run = FALSE to delete.

Usage

```
nlmixr2targets_cache_prune(
  keep = NULL,
  dry_run = TRUE,
  directory = file.path(targets::tar_config_get("store"), "user/nlmixr2")
)
```

Arguments

keep	Character vector of md5 hashes to retain. If NULL (default), the helper auto-discovers reachable hashes from the targets metadata in the current store. Set to character() to treat every cached entry as an orphan.
dry_run	Logical. If TRUE (default), do not delete anything; just return the hashes that would be removed. If FALSE, delete.
directory	Cache directory; same default as nlmixr2targets_cache_status() .

Details

When keep = NULL, the function reads the targets metadata in the current store and treats every value of a built target whose name ends in _object_simple (the hashes produced by [nlmixr_object_simplify\(\)](#)) as reachable. Pass an explicit character vector if you want to override that, e.g. before calling [targets::tar_destroy\(\)](#) yourself.

Value

Invisibly, a character vector of the orphan hashes (those that were removed when dry_run = FALSE, or would be removed otherwise).

See Also

[nlmixr2targets_cache_status\(\)](#).

nlmixr2targets_cache_status

Inspect the nlmixr2targets indirect cache

Description

The indirect cache is a directory of serialized, simplified nlmixrui objects keyed by md5. It lives under <targets store>/user/nlmixr2/ and is normally invisible to `targets::tar_destroy()`; these helpers let you inspect and prune it without losing the rest of the targets store.

Usage

```
nlmixr2targets_cache_status(
  directory = file.path(targets::tar_config_get("store"), "user/nlmixr2")
)
```

Arguments

directory Cache directory. Defaults to <targets::tar_config_get("store")>/user/nlmixr2.

Value

A data frame with one row per cached object and columns hash (md5 file name), size_bytes (file size), and mtime (modification time). Empty data frame with the same columns if the cache directory does not exist or is empty.

See Also

`nlmixr2targets_cache_prune()` for removing orphaned entries.

print.nlmixr2targetsError

Print a failed nlmixr2targets estimation result

Description

When `tar_nlmixr()` or `tar_nlmixr_multimodel()` is called with `error = "continue"` and the estimation step throws, the target stores a lightweight failure sentinel of class `nlmixr2targetsError` instead of an `nlmixr2` fit. This method prints the captured error message. Detect such a sentinel with `inherits(x, "nlmixr2targetsError")` (or the broader `inherits(x, "try-error")`).

Usage

```
## S3 method for class 'nlmixr2targetsError'
print(x, ...)
```

Arguments

`x` A `nlmixr2targetsError` object.

`...` Ignored; present for compatibility with `print()`.

Value

`x`, invisibly.

tar_nlmixr

Generate a set of targets for nlmixr estimation

Description

The targets generated will include the name as the final estimation step, `paste(name, "object_simple", sep = "_")` (e.g. "pheno_object_simple") as the simplified model object, `paste(name, "covariates", sep = "_")` (e.g. "pheno_covariates") as the names of the covariates the model reads, `paste(name, "data_simple", sep = "_")` (e.g. "pheno_data_simple") as the simplified data object, and `paste(name, "fit_simple", sep = "_")` as the fit of the simplified model to the simplified data.

Usage

```
tar_nlmixr(
  name,
  object,
  data,
  est = NULL,
  control = list(),
  table = nlmixr2est::tableControl(),
  env = parent.frame(),
  error = c("stop", "continue"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  library = targets::tar_option_get("library"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  resources_data = targets::tar_option_get("resources"),
  resources_fit = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue")
)

tar_nlmixr_raw(
  name,
  object,
```

```

    data,
    est,
    control,
    table,
    object_simple_name,
    data_simple_name,
    fit_simple_name,
    env,
    error = "stop",
    description = NULL,
    target_settings = tar_nlmixr_collect_target_settings_default(),
    covariates_name = paste(name, "covariates", sep = "_")
)

```

Arguments

name	Symbol, name of the target. In <code>tar_target()</code>, name is an unevaluated symbol, e.g. <code>tar_target(name = data)</code>. In <code>tar_target_raw()</code>, name is a character string, e.g. <code>tar_target_raw(name = "data")</code>. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. <code>tar_target(downstream_target, f(upstream_target))</code> is a target named <code>downstream_target</code> which depends on a target <code>upstream_target</code> and a function <code>f()</code>. In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different case may overwrite a different target. Please ensure all target names have unique names when converted to lower case. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with <code>tar_meta(your_target, seed)</code> and run <code>tar_seed_set()</code> on the result to locally recreate the target's initial RNG state.
object	Fitted object or function specifying the model.
data	nlmixr data
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
table	The output table control object (like <code>'tableControl()'</code>)
env	The environment where the model is setup (not needed for typical use)
error	What should happen if the estimation step throws an error? "stop" (the default) lets the error propagate, halting <code>targets::tar_make()</code> as usual. "continue" catches the error and stores a failure sentinel (an object of class <code>nlmixr2targetsError</code> ,

which also inherits from "try-error") carrying the error message, so a single failed model does not stop the rest of the pipeline. Detect a failed fit with `inherits(fit, "nlmixr2targetsError")` or the broader `inherits(fit, "try-error")`.

format	Optional storage format for the target's return value. With the exception of <code>format = "file"</code> , each target gets a file in <code>_targets/objects</code> , and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices: • "local": file system of the local machine. • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of <code>tar_resources_aws()</code>, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • A character string from <code>tar_repository_cas()</code> for content-addressable storage. Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. As of targets version 1.11.0 and higher, the local file is no longer deleted after the target runs.
library	Character vector of library paths to try when loading packages.
memory	Character of length 1, memory strategy. Possible values: • "auto" (default): equivalent to <code>memory = "transient"</code> in almost all cases. But to avoid superfluous reads from disk, <code>memory = "auto"</code> is equivalent to <code>memory = "persistent"</code> for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has <code>tar_target(name = y, command = x, pattern = map(x))</code>, then <code>tar_target(name = x, command = f(), memory = "auto")</code> will use persistent memory for <code>x</code> in order to avoid rereading all of <code>x</code> for every branch of <code>y</code>. • "transient": the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. • "persistent": the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. `format = "file"` with `repository = "aws"`), the memory option applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection	Logical: TRUE to run <code>base::gc()</code> just before the target runs, in whatever R process it is about to run (which could be a parallel worker). FALSE to omit garbage collection. Numeric values get converted to FALSE. The <code>garbage_collection</code> option in <code>tar_option_set()</code> is independent of the argument of the same name in <code>tar_target()</code> .
deployment	Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html .
resources_data, resources_fit	Object returned by <code>targets::tar_resources()</code> , as for resources in <code>targets::tar_target()</code> . <code>resources_data</code> applies to the <code>_data_simple</code> target. <code>resources_fit</code> applies to every other generated target: the <code>_object_simple</code> , <code>_covariates</code> and <code>_fit_simple</code> targets, the final fit, and the combined list target of <code>tar_nlmixr_multimodel()</code> . See "Running the generated targets on a remote worker".
storage	Character string to control when the output of the target is saved to storage. Only relevant when using targets with parallel workers (https://books.ropensci.org/targets/crew.html). Must be one of the following values: • "worker" (default): the worker saves/uploads the value. • "main": the target's return value is sent back to the host machine and saved/uploaded locally. • "none": targets makes no attempt to save the result of the target to storage in the location where targets expects it to be. Saving to storage is the responsibility of the user. Use with caution.
retrieval	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using targets with parallel workers (https://books.ropensci.org/targets/crew.html). Must be one of the following values: • "auto" (default): equivalent to <code>retrieval = "worker"</code> in almost all cases. But to avoid unnecessary reads from disk, <code>retrieval = "auto"</code> is equivalent to <code>retrieval = "main"</code> for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has <code>tar_target(x, command = f())</code>, then <code>tar_target(y, command = x, pattern = map(x), retrieval = "auto")</code> will use "main" retrieval in order to avoid rereading all of <code>x</code> for every branch of <code>y</code>. • "worker": the worker loads the target's dependencies. • "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • "none": targets makes no attempt to load its dependencies. With <code>retrieval = "none"</code>, loading dependencies is the responsibility of the user. Use with caution.
cue	An optional object from <code>tar_cue()</code> to customize the rules that decide whether the target is up to date.

object_simple_name, data_simple_name, fit_simple_name	target names to use for the simplified object, simplified data, fit of the simplified object with the simplified data, and fit with the original data re-inserted.
description	Human-readable name for the model, announced by the estimation target when it starts to run. NULL (the default) announces nothing.
target_settings	Named list of the <code>targets::tar_target()</code> settings <code>tar_nlmixr()</code> forwards, as built by <code>tar_nlmixr()</code> from its own arguments: one element per forwarded argument, with <code>resources_data</code> and <code>resources_fit</code> in place of <code>resources</code> . Defaults to those arguments' own defaults, so calling <code>tar_nlmixr_raw()</code> directly behaves the same as before this argument existed.
covariates_name	target name to use for the names of the covariates the simplified object reads. Defaults to <code>paste(name, "covariates", sep = "_")</code> , which is the name <code>tar_nlmixr()</code> uses.

Details

For the way that the objects are simplified, see `nlmixr_object_simplify()` and `nlmixr_data_simplify()`. To see how to write initial conditions to work with targets, see `nlmixr_object_simplify()`.

The simplified data keep the columns that the estimation method reads. For most methods that is the standard event columns plus the covariates named in the model; `est = "vae"` additionally searches subject-constant data columns during its automated covariate selection, so those candidate columns are kept, too (see `nlmixr_data_simplify()`). Changes to any kept column invalidate the cached fit; changes to dropped columns do not. The simplified data depend on the model only through the `_covariates` target, so an edit to the model that leaves its covariates unchanged does not rebuild them.

`table` does two jobs: `table$keep` names data columns to carry through to the simplified data, and a non-default `tableControl()` is handed to `nlmixr2est::nlmixr()` so the residual/table step honors it. Left at its default, `table` is omitted from the generated command entirely. That keeps the command byte-identical to what earlier versions produced, so upgrading does not re-run cached fits, and it preserves the merge `nlmixr2est::nlmixr()` performs of the table settings carried on `ui$meta` (`addDosing`, `subsetNonmem`, `cores`, `keep`, `drop`), which happens only when its own `table` argument is missing. Supplying any other `tableControl()` changes the command and re-runs the fit, as a changed control would.

Value

A list of targets for the model simplification, data simplification, and model estimation.

Functions

- `tar_nlmixr_raw()`: An internal function to generate the targets

Arguments not forwarded to `nlmixr2est`

`nlmixr()`: The generated estimation target calls `nlmixr2est::nlmixr()` with `object`, `data`, `est`, `control`, and – when you supply one – `table`. Its three remaining arguments are deliberately left out:

- ... is inert. Each `nlmixr2est::nlmixr()` method captures it with `match.call(expand.dots = TRUE)` and then dispatches to `nlmixr2Est0()` without it, so nothing passed through ... reaches an estimator. There is nothing to forward.
- `save` is accepted by those methods but not acted on by any of them. Even if it were, it writes an RDS copy of the fit to the working directory; the `targets` store already persists every fit, so the copy would be a duplicate produced as an untracked side effect of a target.
- `envir` is the environment `nlmixr2est` passes to `rxode2::udfEnvSet()` to resolve R user-defined functions, and uses to evaluate back-transformation expressions. `nlmixr2targets` leaves it at its default, which resolves to a `nlmixr2targets` internal frame – objects defined in your `_targets.R` are not visible from there. A model that calls an R user-defined function may therefore fail to resolve it; please report it if you hit this.

Arguments not forwarded to targets

`:tar_target()`: `format`, `repository`, `library`, `memory`, `garbage_collection`, `deployment`, `storage`, `retrieval` and `cue` are passed to every generated target. `resources` is split in two: `resources_data` goes to the `_data_simple` target and `resources_fit` to every other target. All of them default exactly as `targets::tar_target()` defaults them (both `resources_*` to `targets::tar_option_get("resources")`), so a `targets::tar_option_set()` earlier in `_targets.R` reaches these targets like any other. The rest are left out on purpose:

- `command`, `deps` and `string` describe the work a target does, and that is what this function exists to write. `string` in particular is already used to keep a renamed model from re-running its fit.
- `pattern` and `iteration` configure dynamic branching. The generated targets are a fixed set of five per model, and `tar_nlmixr_multimodel()` already provides the many-models case, so there is nothing to branch over.
- `tidy_eval` controls `!!` interpolation of a command written by the caller. These commands are assembled with `substitute()` from arguments that are already captured unevaluated, so it has nothing to act on.
- `packages` is chosen by the generator. Each of the five targets per model loads `nlmixr2targets`, whose functions its command calls, and `nlmixr2est`, so that un-namespaced functions in `object`, `control` or `table` (such as `ini()`, `saemControl()` or `tableControl()`) evaluate on a worker that has attached nothing else. The combined list target of `tar_nlmixr_multimodel()` loads no package. Overriding `packages` would break those choices silently. Use `library` to point at a different package library instead. Any other package that `object`, `data`, `control` or `table` needs, such as one set with `targets::tar_option_set(packages = )`, is not declared for these targets, so a worker does not load it; call its functions with `pkg::fun()`.
- `priority` was deprecated in `targets` 1.10.1.9013 (2025-04-08); its scheduler no longer honours user priorities, so forwarding it would only produce a deprecation warning per generated target.
- `error` and `description` are already taken by arguments of this function that mean something else. `error` here decides whether a failed fit becomes a sentinel rather than how `targets` treats a failed target, and `description` names the model in the message the estimation target prints. Set the `targets` versions with `targets::tar_option_set()`.

Running the generated targets on a remote worker

The simplified model is written to `file.path(tar_config_get("store"), "user/nlmixr2")` by the `_object_simple` target and read back from there by the `_covariates` and `estimation(_fit_simple)` targets. That is a path, not a value passed between targets, so the three must agree on it: sending only the estimation target to a worker whose filesystem does not carry the same store leaves it unable to find the model. `resources_fit` covers all three, so leave it as one set of resources that runs them together, or put the store somewhere every worker reaches.

The `_data_simple` target reads neither that directory nor the model: it receives the covariate names as the value of the `_covariates` target. It can therefore run anywhere that can evaluate data, which is what `resources_data` is for – for example, preparing the data on a small local controller while the fits run on a larger one. `deployment` is not split, so it still applies to every generated target.

Side effects

When the user's model function body contains `cmt(0) <- value` inside a `model({...})` block, `tar_nlmixr()` rewrites those lines to `cmt(initial) <- value` directly in the function's binding in `env` so that targets' static analysis (which walks every function in `env` via `codetools::findGlobals()`) accepts the model. The rewrite is reversed at evaluation time, so fitting and downstream behaviour are unchanged. The user-visible consequence is that `body(my_model)` at the REPL after a call to `tar_nlmixr()` will show `cmt(initial)` rather than the originally-written `cmt(0)`.

Manual `cmt(initial) <- value` written by the user is also accepted, but it is a `nlmixr2` targets-only workaround: bare `nlmixr2` does not understand the `cmt(initial)` form, so a model function written that way only fits when routed through `tar_nlmixr()` (or `tar_nlmixr_multimodel()`).

See Also

[tar_nlmixr_multimodel\(\)](#) for fitting many models against one dataset.

Examples

```
pheno <- function() {
  ini({
    lcl <- log(0.008); label("Typical value of clearance")
    lvc <- log(0.6); label("Typical value of volume of distribution")
    etalcl + etalvc ~ c(1,
                      0.01, 1)
    cpaddSd <- 0.1; label("residual variability")
  })
  model({
    cl <- exp(lcl + etalcl)
    vc <- exp(lvc + etalvc)
    kel <- cl / vc
    d / dt(central) <- -kel * central
    cp <- central / vc
    cp ~ add(cpaddSd)
  })
}

# Build the five targets that estimate `pheno`. `data` and `est` are
# captured as expressions, so this just returns the target list; the
```

```
# estimation step runs only when you call `targets::tar_make()` from a
# project whose targets store you have configured (for example, with
# `targets::tar_config_set(store = file.path(tempdir(), "_targets"))`
# or by running inside a project directory you own).
tar_nlmixr(
  name = pheno_model,
  object = pheno,
  data = nlmixr2data::pheno_sd,
  est = "saem"
)
```

`tar_nlmixr_multimodel` *Generate a list of models based on a single dataset and estimation method*

Description

Generate a list of models based on a single dataset and estimation method

Usage

```
tar_nlmixr_multimodel(
  name,
  ...,
  data,
  est,
  control = list(),
  table = nlmixr2est::tableControl(),
  env = parent.frame(),
  error = c("stop", "continue"),
  format = targets::tar_option_get("format"),
  repository = targets::tar_option_get("repository"),
  library = targets::tar_option_get("library"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  resources_data = targets::tar_option_get("resources"),
  resources_fit = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue")
)
```

Arguments

<code>name</code>	Symbol, name of the target. In <code>tar_target()</code> , <code>name</code> is an unevaluated symbol, e.g. <code>tar_target(name = data)</code> . In <code>tar_target_raw()</code> , <code>name</code> is a character string, e.g. <code>tar_target_raw(name = "data")</code> .
-------------------	--

A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`.

In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different case may overwrite a different target. Please ensure all target names have unique names when converted to lower case.

In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.

...	Named arguments with the format "Model description" = modelFunction
data	nlmixr data
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
table	The output table control object (like <code>'tableControl()'</code>)
env	The environment where the model is setup (not needed for typical use)
error	What should happen if the estimation step throws an error? "stop" (the default) lets the error propagate, halting <code>targets::tar_make()</code> as usual. "continue" catches the error and stores a failure sentinel (an object of class <code>nlmixr2targetsError</code> , which also inherits from "try-error") carrying the error message, so a single failed model does not stop the rest of the pipeline. Detect a failed fit with <code>inherits(fit, "nlmixr2targetsError")</code> or the broader <code>inherits(fit, "try-error")</code> .
format	Optional storage format for the target's return value. With the exception of <code>format = "file"</code> , each target gets a file in <code>_targets/objects</code> , and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
repository	Character of length 1, remote repository for target storage. Choices: • "local": file system of the local machine. • "aws": Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the <code>endpoint</code> argument of <code>tar_resources_aws()</code>, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions. • "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

- A character string from `tar_repository_cas()` for content-addressable storage.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. As of targets version 1.11.0 and higher, the local file is no longer deleted after the target runs.

library

Character vector of library paths to try when loading packages.

memory

Character of length 1, memory strategy. Possible values:

- "auto" (default): equivalent to memory = "transient" in almost all cases. But to avoid superfluous reads from disk, memory = "auto" is equivalent to memory = "persistent" for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has `tar_target(name = y, command = x, pattern = map(x))`, then `tar_target(name = x, command = f(), memory = "auto")` will use persistent memory for x in order to avoid rereading all of x for every branch of y.
- "transient": the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- "persistent": the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. format = "file" with repository = "aws"), the memory option applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical: TRUE to run `base::gc()` just before the target runs, in whatever R process it is about to run (which could be a parallel worker). FALSE to omit garbage collection. Numeric values get converted to FALSE. The garbage_collection option in `tar_option_set()` is independent of the argument of the same name in `tar_target()`.

deployment

Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit <https://books.ropensci.org/targets/crew.html>.

resources_data, resources_fit

Object returned by `targets::tar_resources()`, as for resources in `targets::tar_target()`. `resources_data` applies to the `_data_simple` target. `resources_fit` applies to every other generated target: the `_object_simple`, `_covariates` and `_fit_simple` targets, the final fit, and the combined list target of `tar_nlmixr_multimodel()`. See "Running the generated targets on a remote worker".

storage

Character string to control when the output of the target is saved to storage. Only relevant when using targets with parallel workers (<https://books.ropensci.org/targets/crew.html>). Must be one of the following values:

	• "worker" (default): the worker saves/uploads the value. • "main": the target's return value is sent back to the host machine and saved/uploaded locally. • "none": targets makes no attempt to save the result of the target to storage in the location where targets expects it to be. Saving to storage is the responsibility of the user. Use with caution.
retrieval	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using targets with parallel workers (https://books.ropensci.org/targets/crew.html). Must be one of the following values: • "auto" (default): equivalent to retrieval = "worker" in almost all cases. But to avoid unnecessary reads from disk, retrieval = "auto" is equivalent to retrieval = "main" for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has <code>tar_target(x, command = f())</code>, then <code>tar_target(y, command = x, pattern = map(x), retrieval = "auto")</code> will use "main" retrieval in order to avoid rereading all of x for every branch of y. • "worker": the worker loads the target's dependencies. • "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • "none": targets makes no attempt to load its dependencies. With retrieval = "none", loading dependencies is the responsibility of the user. Use with caution.
cue	An optional object from <code>tar_cue()</code> to customize the rules that decide whether the target is up to date.

Value

A list of targets for the model simplification, data simplification, and model estimation.

See Also

[tar_nlmixr\(\)](#) for fitting a single model.

Examples

```
pheno <- function() {
  ini({
    lcl <- log(0.008); label("Typical value of clearance")
    lvc <- log(0.6); label("Typical value of volume of distribution")
    etalcl + etalvc ~ c(1,
                      0.01, 1)
    cpaddSd <- 0.1; label("residual variability")
  })
  model({
    cl <- exp(lcl + etalcl)
    vc <- exp(lvc + etalvc)
  })
}
```

```

      kel <- cl / vc
      d / dt(central) <- -kel * central
      cp <- central / vc
      cp ~ add(cpaddSd)
    })
  }
  pheno2 <- function() {
    ini({
      lcl <- log(0.008); label("Typical value of clearance")
      lvc <- log(0.6); label("Typical value of volume of distribution")
      etalcl + etalvc ~ c(2,
                        0.01, 2)
      cpaddSd <- 3.0; label("residual variability")
    })
    model({
      cl <- exp(lcl + etalcl)
      vc <- exp(lvc + etalvc)
      kel <- cl / vc
      d / dt(central) <- -kel * central
      cp <- central / vc
      cp ~ add(cpaddSd)
    })
  }

  # Build the per-model target chains plus the combined list target.
  # Estimation runs only when `targets::tar_make()` is invoked from a
  # project whose store you have configured (see `?tar_nlmixr` for one
  # tempdir-based setup).
  tar_nlmixr_multimodel(
    name = all_models,
    data = nlmixr2data::pheno_sd,
    est = "saem",
    "Base model" = pheno,
    "Alternative residual error" = pheno2
  )

```

tar_nlmixr_multimodel_has_self_reference

Does the model list refer to another model in the model list?

Description

Does the model list refer to another model in the model list?

Usage

```
tar_nlmixr_multimodel_has_self_reference(model_list, name)
```

```
tar_nlmixr_multimodel_has_self_reference_single(model, name)
```

Arguments

model_list	A named list of calls for model targets to be created
name	Symbol, name of the target. In <code>tar_target()</code>, name is an unevaluated symbol, e.g. <code>tar_target(name = data)</code>. In <code>tar_target_raw()</code>, name is a character string, e.g. <code>tar_target_raw(name = "data")</code>. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. <code>tar_target(downstream_target, f(upstream_target))</code> is a target named <code>downstream_target</code> which depends on a target <code>upstream_target</code> and a function <code>f()</code>. In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different case may overwrite a different target. Please ensure all target names have unique names when converted to lower case. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with <code>tar_meta(your_target, seed)</code> and run <code>tar_seed_set()</code> on the result to locally recreate the target's initial RNG state.
model	A single model call for the model target to be created

Value

A logical vector the same length as `model_list` indicating if the model is self-referential to another model in the list

Functions

- `tar_nlmixr_multimodel_has_self_reference_single()`: A helper function to look at each call for each model separately

```
tar_nlmixr_multimodel_parse
```

*Generate nlmixr multimodel target set for all models in one call to
tar_nlmixr_multimodel()*

Description

Generate nlmixr multimodel target set for all models in one call to `tar_nlmixr_multimodel()`

Usage

```
tar_nlmixr_multimodel_parse(
  name,
  data,
  est,
  control,
  table,
  model_list,
  env,
  error = "stop",
  target_settings = tar_nlmixr_collect_target_settings_default()
)
```

Arguments

name	Symbol, name of the target. In <code>tar_target()</code>, name is an unevaluated symbol, e.g. <code>tar_target(name = data)</code>. In <code>tar_target_raw()</code>, name is a character string, e.g. <code>tar_target_raw(name = "data")</code>. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. <code>tar_target(downstream_target, f(upstream_target))</code> is a target named <code>downstream_target</code> which depends on a target <code>upstream_target</code> and a function <code>f()</code>. In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different case may overwrite a different target. Please ensure all target names have unique names when converted to lower case. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with <code>tar_meta(your_target, seed)</code> and run <code>tar_seed_set()</code> on the result to locally recreate the target's initial RNG state.
data	nlmixr data
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
table	The output table control object (like <code>'tableControl()'</code>)
model_list	A named list of calls for model targets to be created
env	The environment where the model is setup (not needed for typical use)
error	What should happen if the estimation step throws an error? "stop" (the default) lets the error propagate, halting <code>targets::tar_make()</code> as usual. "continue" catches the error and stores a failure sentinel (an object of class <code>nlmixr2targetsError</code> ,

which also inherits from "try-error") carrying the error message, so a single failed model does not stop the rest of the pipeline. Detect a failed fit with `inherits(fit, "nlmixr2targetsError")` or the broader `inherits(fit, "try-error")`.

`target_settings`

Named list of the `targets::tar_target()` settings `tar_nlmixr()` forwards, as built by `tar_nlmixr()` from its own arguments: one element per forwarded argument, with `resources_data` and `resources_fit` in place of `resources`. Defaults to those arguments' own defaults, so calling `tar_nlmixr_raw()` directly behaves the same as before this argument existed.

`tar_nlmixr_multimodel_prep`

Generate the target sets for every model in a model list

Description

The name of each `model_list` element becomes that model's description, so the estimation target can announce which model it is running.

Usage

```
tar_nlmixr_multimodel_prep(
  model_list,
  name,
  data,
  est,
  control,
  table,
  env,
  error = "stop",
  target_settings = tar_nlmixr_collect_target_settings_default()
)
```

Arguments

<code>model_list</code>	A named list of calls for model targets to be created
<code>name</code>	Symbol, name of the target. In <code>tar_target()</code>, <code>name</code> is an unevaluated symbol, e.g. <code>tar_target(name = data)</code>. In <code>tar_target_raw()</code>, <code>name</code> is a character string, e.g. <code>tar_target_raw(name = "data")</code>. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. <code>tar_target(downstream_target, f(upstream_target))</code> is a target named <code>downstream_target</code> which depends on a target <code>upstream_target</code> and a function <code>f()</code>. In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different

case may overwrite a different target. Please ensure all target names have unique names when converted to lower case.

In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.

data	nlmixr data
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
table	The output table control object (like <code>'tableControl()'</code>)
env	The environment where the model is setup (not needed for typical use)
error	What should happen if the estimation step throws an error? "stop" (the default) lets the error propagate, halting <code>targets::tar_make()</code> as usual. "continue" catches the error and stores a failure sentinel (an object of class <code>nlmixr2targetsError</code> , which also inherits from "try-error") carrying the error message, so a single failed model does not stop the rest of the pipeline. Detect a failed fit with <code>inherits(fit, "nlmixr2targetsError")</code> or the broader <code>inherits(fit, "try-error")</code> .
target_settings	Named list of the <code>targets::tar_target()</code> settings <code>tar_nlmixr()</code> forwards, as built by <code>tar_nlmixr()</code> from its own arguments: one element per forwarded argument, with <code>resources_data</code> and <code>resources_fit</code> in place of <code>resources</code> . Defaults to those arguments' own defaults, so calling <code>tar_nlmixr_raw()</code> directly behaves the same as before this argument existed.

Details

`data`, `est`, `control`, and `table` are unevaluated language captured by `substitute()` in `tar_nlmixr_multimodel()` and must reach `tar_nlmixr_multimodel_single()` that way: `data` is typically the name of an upstream target, which has no value while the pipeline is being built. They are handed over directly rather than through an apply-family function's `...`, because `mapply()/Map()` splice their `MoreArgs` values into a constructed call and evaluate them.

`tar_nlmixr_multimodel_single`

Generate a single nlmixr multimodel target set for one model

Description

Generate a single nlmixr multimodel target set for one model

Usage

```
tar_nlmixr_multimodel_single(
  object,
  name,
  data,
  est,
  control,
  table,
  env,
  error = "stop",
  description = NULL,
  target_settings = tar_nlmixr_collect_target_settings_default()
)
```

Arguments

object	Fitted object or function specifying the model.
name	Symbol, name of the target. In <code>tar_target()</code>, name is an unevaluated symbol, e.g. <code>tar_target(name = data)</code>. In <code>tar_target_raw()</code>, name is a character string, e.g. <code>tar_target_raw(name = "data")</code>. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. <code>tar_target(downstream_target, f(upstream_target))</code> is a target named <code>downstream_target</code> which depends on a target <code>upstream_target</code> and a function <code>f()</code>. In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different case may overwrite a different target. Please ensure all target names have unique names when converted to lower case. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with <code>tar_meta(your_target, seed)</code> and run <code>tar_seed_set()</code> on the result to locally recreate the target's initial RNG state.
data	nlmixr data
est	estimation method (all methods are shown by <code>'nlmixr2AllEst()'</code>). Methods can be added for other tools
control	The estimation control object. These are expected to be different for each type of estimation method
table	The output table control object (like <code>'tableControl()'</code>)
env	The environment where the model is setup (not needed for typical use)
error	What should happen if the estimation step throws an error? "stop" (the default) lets the error propagate, halting targets::tar_make() as usual. "continue" catches the error and stores a failure sentinel (an object of class <code>nlmixr2targetsError</code> ,

which also inherits from "try-error") carrying the error message, so a single failed model does not stop the rest of the pipeline. Detect a failed fit with `inherits(fit, "nlmixr2targetsError")` or the broader `inherits(fit, "try-error")`.

description Human-readable name for the model, announced by the estimation target when it starts to run. NULL (the default) announces nothing.

target_settings Named list of the `targets::tar_target()` settings `tar_nlmixr()` forwards, as built by `tar_nlmixr()` from its own arguments: one element per forwarded argument, with `resources_data` and `resources_fit` in place of `resources`. Defaults to those arguments' own defaults, so calling `tar_nlmixr_raw()` directly behaves the same as before this argument existed.

Index

* Internal

assign_origData, 2
nlmixr_object_zero_initial_eval, 7
tar_nlmixr_multimodel_has_self_reference, 23
tar_nlmixr_multimodel_parse, 24
tar_nlmixr_multimodel_prep, 26
tar_nlmixr_multimodel_single, 27

* Simplifiers

nlmixr_data_simplify, 3
nlmixr_object_complicate, 4
nlmixr_object_covariates, 5
nlmixr_object_simplify, 6

assign_origData, 2
assign_origData(), 4, 5

nlmixr2_indirect, 8
nlmixr2est::nlmixr(), 9, 16
nlmixr2targets_cache_prune, 10
nlmixr2targets_cache_prune(), 11
nlmixr2targets_cache_status, 11
nlmixr2targets_cache_status(), 10
nlmixr_data_simplify, 3
nlmixr_data_simplify(), 4–7
nlmixr_object_complicate, 4
nlmixr_object_complicate(), 4, 6, 7
nlmixr_object_covariates, 5
nlmixr_object_covariates(), 3–5, 7
nlmixr_object_simplify, 6
nlmixr_object_simplify(), 4–6, 9, 10
nlmixr_object_zero_initial_eval, 7

print(), 12
print.nlmixr2targetsError, 11

tar_nlmixr, 12
tar_nlmixr(), 4–6, 8, 9, 11, 22
tar_nlmixr_multimodel, 19
tar_nlmixr_multimodel(), 11, 15, 17, 18, 21

tar_nlmixr_multimodel_has_self_reference, 23
tar_nlmixr_multimodel_has_self_reference_single (tar_nlmixr_multimodel_has_self_reference), 23
tar_nlmixr_multimodel_parse, 24
tar_nlmixr_multimodel_prep, 26
tar_nlmixr_multimodel_single, 27
tar_nlmixr_raw(tar_nlmixr), 12
tar_option_set(), 15, 21
tar_repository_cas(), 14, 21
tar_resources_aws(), 14, 20
tar_seed_set(), 13, 20, 24, 25, 27, 28
tar_target(), 13, 15, 19, 21, 24–26, 28
tar_target_raw(), 13, 19, 24–26, 28
targets::tar_destroy(), 10, 11
targets::tar_resources(), 15, 21
targets::tar_target(), 15–17, 21, 26, 27, 29