Package: jheem2 (via r-universe)

November 13, 2026

Type Package

Title Code to set up and run simulations using the JHEEM Engine

Version 1.13.0

Date 2026-11-13

Author Anthony Fojo

Maintainer Anthony Fojo <anthony.fojo@jhmi.edu>

Description The engine for the JHEEM (Joint HIV Epidemiology and Economic Model) family of compartmental models of HIV and STI transmission. Provides tools to specify versioned model structures, compile and run them as differential-equation simulations, manage surveillance data, map between data ontologies, define interventions, and calibrate models to data via likelihoods.

License GPL (>= 2)

Imports Rcpp (>= 1.0.10), R6, bayesian.simulations, codetools, deSolve, distributions, ggnewscale, ggplot2, ggsci, locations, methods, mvtnorm, plotly, reshape2, scales, stats, stringr, tools, ungroup, utils

Remotes tfojo1/bayesian.simulations, tfojo1/distributions, tfojo1/locations

LinkingTo Rcpp

RoxygenNote 7.3.2

Depends R (>= 2.10)

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

Repository https://cipher-epi.r-universe.dev

Date/Publication 2026-11-13 22:14:00 UTC

RemoteUrl https://github.com/CIPHER-Epi/jheem2

RemoteRef v1.13.0

RemoteSha c1c2d3964c49f72da45a3a6595fcdf1203f65b5b

Contents

[.ontology . . . 7
[[<-.ontology . . . 7
[<-.ontology . . . 8
$<-.ontology . . . 8
add.sub.version . . . 9
anchor.intervention.effect.location.and.version . . . 9
apply.ontology.mapping . . . 10
are.simulations.present . . . 11
array.access . . . 11
as.list.ontology . . . 12
as.ontology . . . 13
assemble.mcmc.from.calibration . . . 13
assemble.simulations.from.calibration . . . 14
assemble.transmuted.simulations . . . 14
burn.simset . . . 15
cache.mcmc.summary . . . 15
can.transmute.versions . . . 16
clear.all.managers . . . 16
clear.calibration.cache . . . 17
clear.calibrations . . . 17
clear.interventions . . . 17
clear.ontology.error.debug.info . . . 18
clear.ontology.mappings . . . 18
clear.transmute.calibration.cache . . . 18
clear.versions . . . 19
collapse.array.according.to.indices . . . 19
combine.ontology.mappings . . . 20
compare.sims . . . 20
compute.likelihood . . . 21
compute.likelihood.piecewise . . . 21
copy.calibration.info . . . 22
copy.data.manager . . . 23
copy.simulation.set . . . 23
create.basic.likelihood.instructions . . . 24
create.basic.likelihood.instructions.with.included.multiplier . . . 26
create.basic.likelihood.instructions.with.specified.outcome . . . 29
create.basic.ratio.likelihood.instructions . . . 31
create.bernoulli.likelihood.instructions . . . 33
create.custom.likelihood.instructions . . . 34
create.data.binding . . . 34
create.data.manager . . . 35
create.data.manager.alias . . . 35

create.ifelse.likelihood.instructions . . . 36
create.intervention . . . 36
create.intervention.effect . . . 37
create.jheem.engine . . . 38
create.jheem.specification . . . 39
create.jheem.transmuter . . . 41
create.likelihood.weights . . . 42
create.linear.functional.form . . . 42
create.linear.spline.functional.form . . . 44
create.location.based.ifelse.likelihood.instructions . . . 45
create.log.linear.functional.form . . . 46
create.logistic.linear.functional.form . . . 47
create.logistic.spline.functional.form . . . 48
create.logistic.tail.functional.form . . . 49
create.model.foreground . . . 51
create.monotonic.criteria.based.intervention . . . 51
create.monotonic.criterion . . . 52
create.natural.spline.functional.form . . . 54
create.nested.proportion.likelihood.instructions . . . 55
create.nested.proportion.likelihood.instructions.with.included.multiplier . . . 59
create.ontology.mapping . . . 63
create.other.catchall.ontology.mapping . . . 64
create.outcome.location.mapping . . . 65
create.outcome.metadata . . . 65
create.simset.collection . . . 66
create.solver.metadata . . . 67
create.static.functional.form . . . 67
create.style.manager . . . 68
create.target.population . . . 69
create.time.lagged.comparison.likelihood.instructions . . . 70
create.time.lagged.comparison.likelihood.instructions.with.included.multiplier . . . 72
create.time.lagged.comparison.nested.proportion.likelihood.instructions . . . 75
declare.data.managers . . . 79
diff.target.populations . . . 79
disable.jheem.solver.tracking . . . 80
distribute.dimension.values . . . 80
enable.jheem.solver.tracking . . . 81
examine.get.mappings.to.align.ontologies . . . 81
examine.get.ontology.mapping . . . 82
examine.recent.failed.ontology.mapping . . . 83
execute.plotly.plot . . . 83
execute.simplot . . . 84
expand.array . . . 85
extract.last.simulation.from.calibration . . . 85
find.ontology.mapping . . . 86
get.calibration.progress . . . 87
get.collapse.array.indices . . . 88
get.cumulative.age.counts.smoother . . . 88

get.data.binding . . . 89
get.data.binding.identity . . . 89
get.data.inputs . . . 90
get.data.manager . . . 91
get.data.manager.provenance . . . 91
get.data.outcome.pretty.names . . . 92
get.data.source.full.names . . . 93
get.default.data.manager . . . 93
get.default.style.manager . . . 94
get.expand.array.indices . . . 94
get.expand.then.collapse.array.indices . . . 94
get.identity.ontology.mapping . . . 95
get.jheem.root.directory . . . 95
get.jheem.solver.tracked.info . . . 96
get.locations.with.data . . . 96
get.mappings.to.align.ontologies . . . 97
get.mcmc.mixing.statistic . . . 97
get.ontology.error.debug.info . . . 98
get.ontology.mapping . . . 98
get.ontology.mapping.matrix . . . 99
get.p.bias.estimates . . . 100
get.prior.versions . . . 101
get.registered.ontology . . . 102
get.simset.data . . . 102
get.simset.filename . . . 103
get.simset.filename.and.dir . . . 104
get.simset.params . . . 104
get.simulation.metadata . . . 105
get.specification.for.version . . . 106
get.specification.metadata . . . 106
get.year.bounds.for.outcome . . . 107
import.data . . . 107
incomplete.dimensions . . . 108
incomplete.dimensions.ontology . . . 108
inspect.marginals . . . 109
instantiate.likelihood . . . 110
interpolate . . . 110
intersect.target.populations . . . 111
is.ontology . . . 111
is_complete . . . 112
is_complete.ontology . . . 112
JHEEM . . . 113
jheem2 . . . 118
jheem2-calibration-params . . . 118
jheem2-data-manager-params . . . 119
jheem2-likelihood-params . . . 119
jheem2-params . . . 119
jheem2-simset-params . . . 120

join.interventions . . . 120
join.likelihood.instructions . . . 121
join.outcome.location.mappings . . . 121
join.simulation.sets . . . 122
list.data.managers . . . 122
load.data.manager . . . 123
load.simulation.set . . . 123
make.age.strata.names . . . 124
map.value.ontology . . . 124
mcmc.traceplot . . . 125
names<-.ontology . . . 125
ontology . . . 126
ONTOLOGY.MAPPING . . . 126
outer.join.dim.names . . . 130
parse.age.strata.names . . . 131
parse.simset.filename . . . 131
parse.year.names . . . 131
percentage.cache.complete . . . 132
plot.data.validation . . . 132
plot.simulations . . . 133
prepare.plot . . . 134
print.ontology . . . 135
pull.data . . . 136
put.data . . . 138
put.data.long.form . . . 139
rebuild.data.binding . . . 141
register.aging . . . 142
register.calibrated.parameters.for.version . . . 143
register.calibration.info . . . 144
register.data.manager . . . 145
register.data.metadata.field . . . 145
register.data.ontology . . . 146
register.data.outcome . . . 146
register.data.source . . . 147
register.default.parameter.values . . . 148
register.fixed.model.strata . . . 148
register.initial.population . . . 149
register.model.element . . . 149
register.model.element.values . . . 151
register.model.foreground . . . 152
register.model.mechanism . . . 153
register.model.quantity . . . 154
register.model.quantity.subset . . . 155
register.model.specification . . . 156
register.mortality . . . 156
register.natality . . . 157
register.ontology.mapping . . . 158
register.parent.data.source . . . 159

register.remission . . . 160
register.sampled.parameters.for.version . . . 161
register.set.parameters.for.version . . . 162
register.transition . . . 162
register.transmission . . . 163
register.transmute.calibration.info . . . 165
remove.data . . . 165
rerun.simulations . . . 166
resolve.intervention.effect . . . 167
resolve.ontology.dimension.values . . . 167
restratify.age.counts . . . 168
retrieve.simulation.set . . . 169
run.calibration . . . 169
run.simulations.from.parameters . . . 170
run.transmute.calibration . . . 171
save.simulation.set . . . 171
set.data.binding . . . 172
set.data.release.resolver . . . 172
set.default.data.manager . . . 173
set.default.style.manager . . . 173
set.element.functional.form.alphas.from.parameters . . . 173
set.element.functional.form.from.time . . . 174
set.element.functional.form.future.slope . . . 175
set.element.functional.form.interaction.alphas . . . 176
set.element.functional.form.main.effect.alphas . . . 177
set.element.ramp.times . . . 178
set.element.value . . . 180
set.jheem.root.directory . . . 180
set.up.calibration . . . 181
set.up.transmute.calibration . . . 181
simplot . . . 182
simplot.data.only . . . 184
standardize.age.strata.names . . . 185
STATIC.FUNCTIONAL.FORM . . . 186
subset.simset . . . 187
subset_data . . . 187
thin.simset . . . 188
track.cumulative.outcome . . . 189
track.cumulative.proportion.from.rate . . . 191
track.dynamic.outcome . . . 193
track.integrated.outcome . . . 195
track.point.outcome . . . 198
track.sub.version.outcomes . . . 200
track.transition . . . 200
union.dimension.values . . . 203
union.target.populations . . . 203
update.outcome.keep.dimensions . . . 204
using.data.binding . . . 204

version.has.been.registered 205

Index **206**

`[.ontology` *Subset an ont*

Description

Subset an ont

Usage

```
## S3 method for class 'ontology'
ont[i]
```

Arguments

`ont`	An ontology, as created by `ontology`
`i`	What to subset

Value

An ontology

`[[<-.ontology` *Modify an ontology*

Description

Modify an ontology

Usage

```
## S3 replacement method for class 'ontology'
ont[[i]] <- value
```

Arguments

`ont`	An ontology, as created by `ontology`
`i`	What to subset
`value`	New value to insert

Value

An ontology

[<-.ontology *Modify an ontology*

Description

Modify an ontology

Usage

```
## S3 replacement method for class 'ontology'
ont[i] <- value
```

Arguments

ont	An ontology, as created by ontology
i	What to subset
value	New values to insert

Value

An ontology

$<-.ontology *Modify an ontology*

Description

Modify an ontology

Usage

```
## S3 replacement method for class 'ontology'
ont$i <- value
```

Arguments

ont	An ontology, as created by ontology
i	What to subset
value	New value to insert

Value

An ontology

add.sub.version *Add Sub Version to Specification*

Description

Add Sub Version to Specification

Usage

```
add.sub.version(
  specification,
  sub.version,
  description,
  inherit.outcomes = T,
  can.seed.new.engine = T
)
```

anchor.intervention.effect.location.and.version *Anchor Intervention Effect's Location and Version*

Description

Anchor Intervention Effect's Location and Version

Usage

```
anchor.intervention.effect.location.and.version(
  intervention.effect,
  location,
  specification.metadata,
  error.prefix = ""
)
```

Arguments

intervention.effect	A 'jheem.intervention.effect' object that does not have a location or version set
specification.metadata	A 'jheem.specification.metadata' object for the desired version. Its location need not match the location given as argument to this function.

Value

A new intervention effect that has location and version set

`apply.ontology.mapping` *Apply an Ontology Mapping to Data*

Description

Apply an Ontology Mapping to Data

Usage

```
apply.ontology.mapping(
  ontology.mapping,
  from.arr,
  to.dim.names = NULL,
  fun = "sum",
  na.rm = F,
  error.prefix = ""
)
```

Arguments

`ontology.mapping`	An object of class 'ontology.mapping' - obtained either through register.ontology.mapping, get.ontology.mapping, get.mappings.to.align.ontologies
`from.arr`	An array to transform. Can have any underlying data type, although the function is optimized for numeric data
`to.dim.names`	The dimnames that the resulting transformed data should have (can represent a subset of the dimnames that the ontology mapping knows how to transform). If NULL, the ontology mapping will keep the dimnames in from.arr except for dimensions that are transformed
`fun`	A function used to combine values, or the name of a function. The function must be able to accept any number of arguments (including none). The default, 'sum', is optimized for numeric arrays
`na.rm`	Passed through to fun
`error.prefix`	A character value to be prepended to any error messages thrown by the function

See Also

Other Functions to identify why ontologies fail to map: `get.identity.ontology.mapping()`, `map.value.ontology()`

`are.simulations.present` *Check if Simulations Have Been Saved*

Description

Summarize the progress of a calibration task

Usage

```
are.simulations.present(
  version,
  locations,
  calibration.code,
  interventions,
  n.sim,
  sub.version = NULL,
  root.dir = get.jheem.root.directory()
)
```

Arguments

`version`, `calibration.code`, `locations`	The version, locations, and calibration code for which to gauge progress
`root.dir`	The root directory from which to search for calibrations
`as.pct`	Whether the progress should be shown as a percentage instead of a fraction
`round.to.digits`	How many digits to round progress to

`array.access` *Functions for subsetting an array when the number and order of dimensions is not known at the point of coding. The array.access function performs error checking on the arguments; fast.array.access is optimized for speed but does not check arguments*

Description

Flexibly Subset an Array

Usage

```
array.access(arr, ..., drop = F)

fast.array.access(arr, dimension.values, drop = F)
```

Arguments

arr	The array or matrix to be subsetted. Must have named dimensions and named dimnames set
drop	Pass-through for the drop argument for array access. If set to true, any dimensions in the return value that have length one are omitted (as with standard subsetting). If false, then the dimensions are retained in the array
dimension.values	A named list containing the values of dimensions to subset for. The names of dimension.values should correspond to the names of dimnames. The elements of dimension.values should be character, integer, or logical vectors

Functions

- fast.array.access():

Examples

```
arr = array(1:24, dim=c(age=4, race=3, sex=2),
                dimnames=list(age=c('age1','age2','age3','age4'),
                              race=c('black','hispanic','other'),
                              sex=c('female','male')))

array.access(arr, dimension.values=list(age=1:3, race='black'))
#is equivalent to
arr[1:3, 'black',]
```

---

as.list.ontology *Convert an ontology back to a plain list*

Description

Convert an ontology back to a plain list

Usage

```
## S3 method for class 'ontology'
as.list(ont)
```

Arguments

ont	An ontology as created by ontology

as.ontology *Convert a named list to an ontology*

Description

Convert a named list to an ontology

Usage

```
as.ontology(x, incomplete.dimensions = NULL)
```

Arguments

x	A named list of character vectors or NULL elements

assemble.mcmc.from.calibration *Assemble an MCMC From a Calibration*

Description

Assemble an MCMC From a Calibration

Usage

```
assemble.mcmc.from.calibration(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory("Cannot set up calibration: "),
  allow.incomplete = F,
  chains = NULL
)
```

Arguments

version	The version of the model specification (must have been previously registered)
location	A single character value representing the location for the calibration
chains	

assemble.simulations.from.calibration
Assemble Simulations From A Calibration

Description

Assemble Simulations From A Calibration

Usage

```
assemble.simulations.from.calibration(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory("Cannot set up calibration: "),
  allow.incomplete = F,
  chains = NULL,
  verbose = T
)
```

Arguments

version	The version of the model specification (must have been previously registered)
location	A single character value representing the location for the calibration
chains	

assemble.transmuted.simulations
Assemble Transmuted Simulations

Description

Assemble Transmuted Simulations

Usage

```
assemble.transmuted.simulations(
  transmute.code,
  location,
  n.sim,
  to.version = NULL,
  to.sub.version = NULL,
  chunks = NULL,
  allow.incomplete = F,
  root.dir = get.jheem.root.directory()
)
```

`burn.simset` *Burn Simulations From a Simulation Set*

Description

Subset a simulation set by trimming off a portion of earlier simulations.

Usage

```
burn.simset(simset, keep = NULL)
```

Arguments

`keep`	Either a number of simulations to keep or a fraction to keep (between 0 and 1). The latest simulations will be kept.

Details

Decimal values above 1 are rounded down to the nearest whole number.

`cache.mcmc.summary` *Cache A Summary of an MCMC*

Description

Cache A Summary of an MCMC

Usage

```
cache.mcmc.summary(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory(),
  get.one.set.of.parameters = F,
  throw.error.if.incomplete = T
)
```

can.transmute.versions

Assess Whether Simulations from one Version/Specification can be Transmuted to Another Version

Description

'Transmutation' of a simulation is calculating a set of outcomes for another version without re-running the (expensive) differential equation solver

Usage

```
can.transmute.versions(
  from.version,
  to.version,
  from.sub.version = NULL,
  to.sub.version = NULL,
  print.details = T
)
```

Arguments

from.version, to.version	The versions from and to which we want to transmute
from.sub.version, to.sub.versions	The sub-versions from and to which we want to transmute
print.details	Whether to print the reasons why transmutation cannot happen

clear.all.managers

Clear JHEEM Version, Calibration, Intervention, and Ontology Mappings

Description

Clear JHEEM Version, Calibration, Intervention, and Ontology Mappings

Usage

```
clear.all.managers()
```

clear.calibration.cache *Clear a Calibration Cache*

Description

Clear a Calibration Cache

Usage

```
clear.calibration.cache(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory("Cannot set up calibration: "),
  remove.mcmc.summary = F,
  allow.remove.incomplete = F
)
```

Arguments

version	The version of the model specification (must have been previously registered)
location	A single character value representing the location for the calibration
allow.remove.incomplete	

clear.calibrations *Clear JHEEM Calibrations*

Description

Clear JHEEM Calibrations

Usage

```
clear.calibrations()
```

clear.interventions *Clear JHEEM Interventions*

Description

Clear JHEEM Interventions

Usage

```
clear.interventions()
```

clear.ontology.error.debug.info
Clear Ontology Error Debug Info

Description

Clear Ontology Error Debug Info

Usage

```
clear.ontology.error.debug.info()
```

clear.ontology.mappings
Clear All Registered Ontology Mappings

Description

Clear All Registered Ontology Mappings

Usage

```
clear.ontology.mappings()
```

Details

Clears all registered ontology mappings, such that new mappings can be added

clear.transmute.calibration.cache
Clear a Calibration Cache

Description

Clear a Calibration Cache

Usage

```
clear.transmute.calibration.cache(
  transmute.code,
  location,
  n.sim,
  to.version = NULL,
  to.sub.version = NULL,
  allow.remove.incomplete = F,
  root.dir = get.jheem.root.directory()
)
```

Arguments

location	A single character value representing the location for the calibration
allow.remove.incomplete	

---

clear.versions *Clear JHEEM Versions*

Description

Clear JHEEM Versions

Usage

```
clear.versions()
```

---

collapse.array.according.to.indices *Collapse an Array According to Indices*

Description

Collapse an Array According to Indices

Usage

```
collapse.array.according.to.indices(
  arr,
  small.indices,
  large.indices,
  small.n,
  check.consistency = T
)
```

combine.ontology.mappings *Combine multiple ontology mappings into a single mapping*

Description

Combine multiple ontology mappings into a single mapping

Usage

```
combine.ontology.mappings(..., thorough.validation = T)
```

Arguments

...	Either ontology.mapping objects or lists containing ontology.mapping objects#'

compare.sims *Compare Two Simsets*

Description

Compare how two simsets score on a likelihood.

Usage

```
compare.sims(likelihood, sim1, sim2, piecewise = T, log = F)
```

Arguments

sim1, sim2	'jheem.simulation.set' objects to compare
piecewise	Should they be compared on each likelihood individually, if a joint likelihood is supplied?
log	Whether to use log likelihood

`compute.likelihood` *Compute Likelihood*

Description

Compute Likelihood

Usage

```
compute.likelihood(
  likelihood,
  sim,
  log = T,
  use.optimized.get = F,
  check.consistency = T,
  debug = F
)
```

Arguments

`log`	Whether to use log likelihood
`use.optimized.get`	Should the optimized sim$get be used? The optimized get only works on simsets of the same internal structure as that defined in the specification at the time this likelihood was instantiated, so use it for the MCMC but leave it FALSE for analysis.
`check.consistency`	- Whether to spend time checking to make sure everything is internally consistent. Setting to F is faster, but may generate weird error messages if there are bugs

`compute.likelihood.piecewise`
Compute Joint Likelihood Piecewise

Description

Compute Joint Likelihood Piecewise

Usage

```
compute.likelihood.piecewise(
  likelihood,
  sim,
  log = T,
  use.optimized.get = F,
  check.consistency = T,
  debug = F
)
```

Arguments

log	Whether to use log likelihood
use.optimized.get	Should the optimized sim$get be used? The optimized get only works on simsets of the same internal structure as that defined in the specification at the time this likelihood was instantiated, so use it for the MCMC but leave it FALSE for analysis.
check.consistency	- Whether to spend time checking to make sure everything is internally consistent. Setting to F is faster, but may generate weird error messages if there are bugs

Details

For likelihoods that are not joint likelihoods, this is the same as compute.likelihood()

---

copy.calibration.info *Copy Calibration Info to a New Code*

Description

Copy Calibration Info to a New Code

Usage

```
copy.calibration.info(
  from.code,
  to.code,
  n.iter = NULL,
  n.burn = NULL,
  thin = NULL,
  solver.metadata = NULL
)
```

Arguments

from.code	The code for the the calibration info to copy
to.code	The calibration code to be used for the copied calibration info

---

copy.data.manager *Create a copy of a JHEEM Data Manager*

Description

Create a copy of a JHEEM Data Manager

Usage

```
copy.data.manager(
  data.manager = get.default.data.manager(),
  name,
  description,
  set.as.default = F,
  preserve.original.creation.date = T,
  preserve.original.last.modified.date = T
)
```

Arguments

data.manager	A jheem.data.manager object to be copied
name	The name of the new data manager
description	A short description of the new data manager. If NULL, the description of the copied data manager.
set.as.default	Should the new data manager be set as the default data manager
preserve.original.creation.date, preserve.original.last.modified.date	Should the new data manager keep the copied data manager's creation date/last modified date (T) or set that date to now (F)?

---

copy.simulation.set *Make a copy of a Simulation Set*

Description

Make a copy of a Simulation Set

Usage

```
copy.simulation.set(simset, update.labels = T)
```

Arguments

simset	The simulation.set object to copy
update.labels	Whether to get updated labels from the latest specification (vs the specification as it was when the simulations were created)

`create.basic.likelihood.instructions`
Create JHEEM Basic Likelihood Instructions

Description

Create JHEEM Basic Likelihood Instructions

Usage

```
create.basic.likelihood.instructions(
  outcome.for.data,
  outcome.for.sim,
  dimensions = character(0),
  denominator.dimensions = dimensions,
  dimension.values = NULL,
  levels.of.stratification = NULL,
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
  minimum.error.sd = 0,
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  error.variance.term = NULL,
  error.variance.type = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  name = outcome.for.sim,
  na.rm = NULL
)
```

Arguments

outcome.for.data	A single character vector specifying the outcome that data managers will use to find data.
outcome.for.sim	A single character vector specifying the simulation outcome to use.

`dimensions` A character vector of dimensions, excluding year, from which stratifications will be generated.

`denominator.dimensions`
(Optional) Analogous to 'dimensions' but for the simulation denominator outcome when applicable. Must be a subset of 'dimensions'.

`levels.of.stratification`
An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year` -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year` Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years` (Optional) An integer vector of years to ignore.

`sources.to.use` A character vector of sources from which to pull data.

`minimum.error.sd`
A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`correlation.different.years`
A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`observation.correlation.form`
Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`error.variance.type`, `error.variance.term`
The type of error variance(s) to be used and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`weights` A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`na.rm` Setting this TRUE will pass 'na.rm=T' along to the pull function. If set to NULL, 'na.rm=T' will be used for outcomes with scale 'proportion', 'rate', or 'ratio'.

`measurement.error.coefficient.of.variance`
A single numeric value specifying the coefficient of variance used to build the measurement error covariance matrix.

Details

Error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with an 'error.variance.type' of "cv" and an 'error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like an 'error.variance.type' of 'c("data.sd", "function.sd") and an 'error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

See Also

Other Basic Likelihoods: `create.basic.likelihood.instructions.with.included.multiplier()`, `create.basic.likelihood.instructions.with.specified.outcome()`, `create.time.lagged.comparison.likeliho`
`create.time.lagged.comparison.likelihood.instructions.with.included.multiplier()`

---

create.basic.likelihood.instructions.with.included.multiplier

Create JHEEM Basic Likelihood Instructions With Included Multiplier

---

Description

Create JHEEM Basic Likelihood Instructions With Included Multiplier

Usage

```
create.basic.likelihood.instructions.with.included.multiplier(
  outcome.for.data,
  outcome.for.sim,
  dimensions = character(0),
  denominator.dimensions = dimensions,
  dimension.values = NULL,
  levels.of.stratification = NULL,
```

```
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
  minimum.error.sd = 0,
  included.multiplier,
  included.multiplier.sd,
  included.multiplier.correlation = NULL,
  included.multiplier.correlation.structure = c("compound.symmetry",
    "autoregressive.1")[1],
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  error.variance.term = NULL,
  error.variance.type = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  name = outcome.for.sim,
  na.rm = NULL
)
```

Arguments

`outcome.for.data`	A single character vector specifying the outcome that data managers will use to find data.
`outcome.for.sim`	A single character vector specifying the simulation outcome to use.
`dimensions`	A character vector of dimensions, excluding year, from which stratifications will be generated.
`denominator.dimensions`	(Optional) Analogous to 'dimensions' but for the simulation denominator outcome when applicable. Must be a subset of 'dimensions'.
`levels.of.stratification`	An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.
`from.year`	-Inf or a single integer value specifying the earliest year for which data should be pulled.
`to.year`	Inf or a single integer value specifying the latest year for which data should be pulled.
`omit.years`	(Optional) An integer vector of years to ignore.
`sources.to.use`	A character vector of sources from which to pull data.

`minimum.error.sd`
: A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`included.multiplier`
: The value of the multiplier that has already been used in calculating the simulation value of the outcome (represents the mean of the distribution around the multiplier).

`included.multiplier.sd`
: The standard deviation of the uncertainty around the included multiplier.

`included.multiplier.correlation`
: The correlation between values of the multiplier across different strata and years.

`included.multiplier.correlation.structure`
: Which correlation form to use for values of the multiplier across various strata and years.

`correlation.different.years`
: A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
: A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
: A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`error.variance.type`, `error.variance.term`
: The type of error variance(s) to be used and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`weights`
: A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`na.rm`
: Setting this TRUE will pass 'na.rm=T' along to the pull function. If set to NULL, 'na.rm=T' will be used for outcomes with scale 'proportion', 'rate', or 'ratio'.

Details

Error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s)

of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with an 'error.variance.type' of "cv" and an 'error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like an 'error.variance.type' of 'c("data.sd", "function.sd") and an 'error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

See Also

Other Basic Likelihoods: `create.basic.likelihood.instructions()`, `create.basic.likelihood.instructions.wit`
`create.time.lagged.comparison.likelihood.instructions()`, `create.time.lagged.comparison.likelihood.in`

`create.basic.likelihood.instructions.with.specified.outcome`
Create JHEEM Basic Likelihood Instructions With Specified Outcome

---

Description

Create JHEEM Basic Likelihood Instructions With Specified Outcome

Usage

```
create.basic.likelihood.instructions.with.specified.outcome(
  outcome.for.sim,
  outcome.value,
  from.year,
  to.year,
  omit.years = NULL,
  minimum.error.sd = 0,
  correlation.different.years = 0.5,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  error.variance.term = NULL,
  error.variance.type = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  name = outcome.for.sim,
  na.rm = NULL
)
```

Arguments

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`outcome.value`
: A single, non-NA numeric value that is the value of the outcome at the totals level (i.e., not stratified).

`from.year`
: -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year`
: Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years`
: (Optional) An integer vector of years to ignore.

`minimum.error.sd`
: A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`correlation.different.years`
: A single numeric value specifying the correlation between observations of different years.

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`error.variance.type`, `error.variance.term`
: The type of error variance(s) to be used and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`weights`
: A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`na.rm`
: Setting this TRUE will pass 'na.rm=T' along to the pull function. If set to NULL, 'na.rm=T' will be used for outcomes with scale 'proportion', 'rate', or 'ratio'.

Details

Error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with an 'error.variance.type' of "cv" and an 'error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like an 'error.variance.type' of 'c("data.sd", "function.sd") and an 'error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

See Also

Other Basic Likelihoods: create.basic.likelihood.instructions(), create.basic.likelihood.instructions.wit
create.time.lagged.comparison.likelihood.instructions(), create.time.lagged.comparison.likelihood.in

---

`create.basic.ratio.likelihood.instructions`
Create Basic Ratio Likelihood Instructions

Description

Create Basic Ratio Likelihood Instructions

Usage

```
create.basic.ratio.likelihood.instructions(
  outcome.for.data,
  outcome.for.sim,
  dimensions = character(0),
  denominator.dimensions = dimensions,
  dimension.values = NULL,
  levels.of.stratification = NULL,
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  error.variance.term = NULL,
  error.variance.type = NULL,
  ratio.cv = NULL,
  ratio.correlation = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  name = outcome.for.sim
)
```

Arguments

`outcome.for.data`
: A single character vector specifying the outcome that data managers will use to find data.

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`dimensions`
: A character vector of dimensions, excluding year, from which stratifications will be generated.

`denominator.dimensions`
: (Optional) Analogous to 'dimensions' but for the simulation denominator outcome when applicable. Must be a subset of 'dimensions'.

`levels.of.stratification`
: An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year`
: -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year`
: Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years`
: (Optional) An integer vector of years to ignore.

`sources.to.use`
: A character vector of sources from which to pull data.

`correlation.different.years`
: A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
: A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
: A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`error.variance.type`, `error.variance.term`
: The type of error variance(s) to be used and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`ratio.cv`, `ratio.correlation`
: Covariance and correlation coefficients for a compound symmetry matrix representing...? Must be greater than 0, and between -1 and 1, respectively. If 'ratio.cv' is not NULL, 'ratio.correlation' will default to 0.

`weights` A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year` A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`create.bernoulli.likelihood.instructions`
Create Bernoulli Likelihood Instructions

Description

Create Bernoulli Likelihood Instructions

Usage

```
create.bernoulli.likelihood.instructions(
  outcome.for.sim,
  years,
  probability.decreasing,
  weights,
  dimensions = character(0),
  levels.of.stratification = NULL,
  name = outcome.for.sim
)
```

Arguments

`outcome.for.sim` A single character vector specifying the simulation outcome to use.

`years` A length two numeric vector specifying the baseline year (first) and the year which will be assessed against it (second).

`probability.decreasing` A single numeric value between 0 and 1 specifying probability that the outcome value is lower in the second year than in the first.

`weights` A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`dimensions` A character vector of dimensions, excluding year, from which stratifications will be generated.

`levels.of.stratification` A numeric vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

create.custom.likelihood.instructions *Create Custom Likelihood Instructions*

Description

Create instructions for a likelihood that computes using a custom function.

Usage

```
create.custom.likelihood.instructions(
  name,
  compute.function,
  get.data.function = NULL,
  weights = NULL,
  verbose = F
)
```

Arguments

compute.function	A function that takes only arguments "sim", "log", and "weights", with optional "data" and "debug" arguments.
weights	A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by create.likelihood.weights

Details

All weights used in the compute function should arrive in its "weights" argument in two ways: From the "weights" argument of the likelihood instructions (which apply to these instructions only), and additional weights passed top-down from joint likelihoods that will include this likelihood. So, no weights should be passed in the "data" function, for example, to maintain consistency and accuracy.

create.data.binding *Create a data binding*

Description

A map from names to data managers. Pass it to using.data.binding or set.data.binding to build model objects with it.

Usage

```
create.data.binding(...)
```

Arguments

...	Data managers, each named with the name code uses to get it (letters, digits, '.' and '_', starting with a letter), or a single named list of them

Value

An object of class 'jheem.data.binding'

---

create.data.manager *Create a JHEEM Data Manager*

Description

Create a JHEEM Data Manager

Usage

```
create.data.manager(name, description, set.as.default = F)
```

Arguments

name	The name of the data manager
description	A short description

---

create.data.manager.alias *Make a variable an alias for a bound data manager*

Description

Turns 'variable' into an active binding whose value is `get.data.manager(name)`, so code that reads the variable (for example CENSUS.MANAGER) gets the manager bound to 'name' in the active binding, and the read is recorded. Assigning a data manager to the variable registers it under 'name' (see `register.data.manager`); this is not allowed while a binding other than the session's is active or jheem2 is building under one.

If the variable already holds a data manager, it is registered under 'name' unless another manager already is. Aliasing a variable again to the same name does nothing; aliasing it to a different name is an error.

Usage

```
create.data.manager.alias(variable, name, envir = globalenv())
```

Arguments

variable	The variable's name, for example 'CENSUS.MANAGER'
name	The data manager's name in the binding, for example 'census'
envir	The environment holding the variable

Value

NULL, invisibly

---

create.ifelse.likelihood.instructions

Create If-Else Likelihood Instructions

Description

Create instructions that can try a sequence of likelihood instructions until the first one that works without error.

Usage

```
create.ifelse.likelihood.instructions(..., name = NULL)
```

Arguments

...	'jheem.likelihood.instructions' objects that all have the same 'outcome.for.sim'.
name	Name for the ifelse; will default to

Details

The instructions are attempted in the order that they are supplied.

---

create.intervention *Create an Intervention*

Description

Create an Intervention

Usage

```
create.intervention(
  ...,
  code = NULL,
  name = NULL,
  parameters = NULL,
  parameter.distribution = NULL,
  generate.parameters.function = NULL,
  overwrite.existing.intervention = F
)
```

Arguments

...	Either (a) objects of class 'target.population', (b) objects of class 'intervention.effect', or (d) lists containing only 'target.population' or 'intervention.effect' objects. Must contain at least one 'target.population' object and at least one 'intervention.effect' object
code	A brief (2-30 character) string that will uniquely identify this intervention
parameter.distribution	

Details

Creates a 'jheem.intervention' object where all the intervention effects given in ... apply to all the target populations given in ...

---

create.intervention.effect

Create an Effect for an Intervention to Have on a Model Quantity

---

Description

Create an Effect for an Intervention to Have on a Model Quantity

Usage

```
create.intervention.effect(
  quantity.name,
  start.time,
  effect.values,
  times,
  end.time = Inf,
  apply.effects.as,
  scale,
  allow.values.less.than.otherwise,
  allow.values.greater.than.otherwise,
```

```
  bindings = NULL,
  ...
)
```

Arguments

quantity.name	The name of the quantity, defined in a jheem.model.specification, which an intervention affects
start.time	The time at which the intervention begins to take effect (ie, depart from what the value of the quantity would otherwise have been)
effect.values	The values to apply to the quantity at times. Must be either (1) a numeric vector, (2) a character vector, (3) an expression or call, or (4) a list containing only numeric vectors, character vectors, expressions, or calls
times	The times as which values apply to the quantity.
end.time	The time at which the intervention stops taking effect (ie, returns to what the value of the quantity otherwise would have been)
apply.effects.as	A single character value indicating how values should be applied to the quantity. Choices are (1) 'value' - the quantity takes the given value, (2) 'multiplier' - the value is multiplied by the value the quantity would otherwise have taken, or (3) 'addend' - the value is added to the value the quantity would otherwise have taken.
scale	The scale at which values are applied to the quantity
allow.values.less.than.otherwise, allow.values.greater.than.otherwise	Logical indicators of whether the intervention may cause the value of the quantity to be less/greater than it otherwise would have been
bindings	A named list. The names must correspond to variables on which the intervention effect applies, if it is a variable. The elements of the list must be either (1) numeric values or (2) functions must take only "location" and "specification.metadata" as arguments and return numeric vectors with no NA values
...	Additional arguments to be passed to any bindings which are functions

---

create.jheem.engine *Create an Engine to Run JHEEM Simulations*

Description

Create an Engine to Run JHEEM Simulations

Usage

```
create.jheem.engine(
  version,
  location,
  end.year,
  sub.version = NULL,
  max.run.time.seconds = Inf,
  solver.metadata = NULL
)
```

Arguments

version, location, sub.version	The JHEEM version, sub.version, and location for which to run simulations
end.year	The year to run simulations to
max.run.time.seconds	The maximum amount of time to run simulations before terminating
solver.metadata	A solver.metadata object, as created by create.solver.metadata, specifying what ODE solver to use in running simulations

---

create.jheem.specification *Create a Model Specification for Running the JHEEM*

Description

Create a Model Specification for Running the JHEEM

Usage

```
create.jheem.specification(
  version,
  iteration,
  description,
  start.year,
  parent.version = NULL,
  do.not.inherit.model.quantity.names = character(),
  do.not.inherit.model.outcome.names = character(),
  do.not.inherit.transitions.for.dimension = character(),
  do.not.inherit.components.with.tags = character(),
  compartments.for.infected.only = list(),
  compartments.for.uninfected.only = list(),
  compartments.for.infected.and.uninfected = list(),
  compartments.for.outcomes = list(),
  age.endpoints = NULL,
```

```
  compartment.value.aliases = list(),
  order.locations.by.outcomes = NULL,
  labels = NULL,
  default.solver.metadata = NULL
)
```

Arguments

`version`	A single character value denoting the version
`iteration`	A single character or numeric value denoting the iteration of the model specification for this version
`description`	A short text description of this version
`start.year`	The numeric year at which simulations should start. If NULL inherits the parent specification's start year
`parent.version`	The character version of the specification from which this new specification should inherit. Must have been registered
`do.not.inherit.model.quantity.names`	A vector of names of model.quantities which should NOT be inherited from ancestor specifications
`do.not.inherit.model.outcome.names`	A vector of names of model outcomes which should NOT be inherited from ancestor specifications
`do.not.inherit.transitions.for.dimension`	A vector of names of dimensions for which transitions should NOT be inherited from ancestor specifications
`do.not.inherit.components.with.tags`	A vector of tags for which core components (transmission, natality, mortality, transitions, aging) should NOT be inherited from ancestor specifications
`compartments.for.infected.only`, `compartments.for.uninfected.only`, `compartments.for.infected.and.uninfected`, `compartments.for.outcomes`	Named lists of character vectors F@specifying the compartments for uninfected and infected groups, compartments shared by both, or additional compartments that show in the outcomes. The names of the lists represent dimensions, and the values the compartments for each dimension. Compartments can either be string referencing the compartments themselves, or strings representing aliases passed to compartment.value.aliases
`age.endpoints`	Optional. A numeric vector (with at least two elements) giving the endpoints of the age brackets to use for the 'age' dimension. Results in length(age.endpoints)-1 different brackets, where the first bracket spans age.endpoints[1] (inclusive) to age.endpoints[2] (exclusive), the second bracket spans age.endpoints[2] to age.endpoints[3], etc
`compartment.value.aliases`	A named list representing substitutions to be made into compartment names (both in compartments.for.infected.only, compartments.for.uninfected.only, compartments.for.infected.and.uninfected and in subsequent references in registering model quantities). The names of the list represent what is to be replaced.

The values of the list can be either (1) character vectors that are substituted in place or (2) functions that take parameter 'location' and return a character vector

`order.locations.by.outcomes`
: A character vector of outcome names from a data manager to be referenced in deciding which locations to show if there is a need to prioritize in making plots

`labels`
: A named character vector, representing how compartments should be labeled when making figures/charts, etc. Eg c(msm="MSM", idu="PWID")

`create.jheem.transmuter`
Create a Transmuter

Description

Create a Transmuter

Usage

```
create.jheem.transmuter(
  simulation.set,
  to.version,
  from.year = simulation.set$from.year,
  to.year = simulation.set$to.year,
  to.sub.version = NULL
)
```

Arguments

`simulation.set`
: The simulation.set to transmute

`to.version`, `to.sub.version`
: The version and sub.version to transmute to

Details

A jheem.transmuter can convert simulations from one version/specification to another using its transmute() method

`create.likelihood.weights` *Create Weights to Be Applied in Calculating a JHEEM Likelihood*

Description

Create Weights to Be Applied in Calculating a JHEEM Likelihood

Usage

```
create.likelihood.weights(
  total.weight = 1,
  dimension.values = list(),
  is.recursive = T
)
```

Arguments

`total.weight`	The weight that is applied to all observations in the
`is.recursive`	Should this weight apply to deeper levels of stratification? For example, if F and dimension.values is the default empty list, this weight would only apply to data points that are totals.

`create.linear.functional.form` *Create an Functional Form that is Linear (possibly on a transformed scale)*

Description

Create an Functional Form that is Linear (possibly on a transformed scale)

Usage

```
create.linear.functional.form(
  intercept,
  slope,
  anchor.year,
  link = c("identity", "log", "logit")[1],
  min = NA,
  max = NA,
  intercept.link = NULL,
  intercept.min = NA,
  intercept.max = NA,
  intercept.robust.to.negative = F,
```

```
  slope.link = intercept.link,
  slope.min = NA,
  slope.max = NA,
  slope.robust.to.negative = F,
  parameters.are.on.transformed.scale = T,
  overwrite.parameters.with.alphas = F,
  error.prefix = "Cannot create linear functional.form: "
)
```

Arguments

`intercept`, `slope`
: Arrays or scalar values representing the intercept and slope on the transformed scale

`anchor.year`
: The year at which the functional form evaluates to the intercept

`link`
: The name of the transformation to the scale on which alphas are added to the value. One of "identity", "log", "logit"

`min`, `max`
: The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)

`intercept.link`, `slope.link`
: The names of the transformations to the scale on which alphas are added to calculate the intercept and slope. One of "identity", "log", "logit". If NULL, inferred automatically from the link for the entire functional form

`intercept.min`, `intercept.max`, `slope.min`, `slope.max`
: The min and max values to which intercept and slope can evaluate

`overwrite.parameters.with.alphas`
: Logical indicating whether, when alphas are applied, they OVERWRITE values. If F, alphas are added to the transformed value (on the transformed scale)

`error.prefix`
: String to prepend to any error message in case of an error

`parameters.is.on.transformed.scale`
: Logical indicating whether the intercept and slope are already on the transformed scale at which alphas apply

`intercept.robust.intercept.linkto.negative`, `slope.robust.to.negative`
: Whether intercept and slope should be allowed to be negative, even if a log or logit link is used (this is handled by letting the the alphas apply to the negative of the parameter, then negating it)

See Also

create.static.functional.form, create.log.linear.functional.form, create.logistic.linear.functional.form

`create.linear.spline.functional.form` *Create a Linear Spline Functional Form*

Description

Create a Linear Spline Functional Form

Usage

```
create.linear.spline.functional.form(
  knot.times,
  knot.values,
  link = c("identity", "log", "logit")[1],
  min = NA,
  max = NA,
  knot.link = link,
  knot.min = min,
  knot.max = max,
  knots.are.on.transformed.scale = F,
  overwrite.knot.values.with.alphas = F,
  before.time = NULL,
  before.modifier = NULL,
  after.time = NULL,
  after.modifier = NULL,
  modifiers.apply.to.change = T,
  modifier.link = link,
  after.modifier.increasing.change.link = modifier.link,
  after.modifier.decreasing.change.link = modifier.link,
  before.modifier.increasing.change.link = modifier.link,
  before.modifier.decreasing.change.link = modifier.link,
  modifier.min = min,
  modifier.max = max,
  overwrite.modifiers.with.alphas = F,
  error.prefix = "Cannot create linear spline functional.form: "
)
```

Arguments

`knot.times`	The (numeric) times at which the knots of the spline apply. Must have names set
`knot.values`	Initial values for the knots. Must have names set, matching the names of knot times
`link`	The name of a transformation to the scale at which the SPLINE should apply. The knots are transformed to this scale, the spline is applied, and then the splined values are back-transformed. One of 'identity', 'log', or 'logistic'

`min`, `max` The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)

`knot.link` The name of a transformation to the scale at which the knot values have alphas ADDED. The knots are back-transformed from this scale prior to the spline scale being applied

`knot.min`, `knot.max`
The min and max values that KNOTS can take before being splined (note, the min and max values the functional form can take, after splining, are determined by parameters min and max)

`knots.are.on.transformed.scale`
Logical indicating whether the knots are already on the transformed scale (ie knot.link has been applied)

`before.time`, `after.time`
The times at which the knots calculated from before.modifier and after.modifier apply, respectively

`before.modifier`, `after.modifier`
If either of these is set, an additional knot is added to the specified knots. For before.modifier, this knot is calculated as before.modifier added or multiplier by the value of the first knot (on the model scale); after.modifier is added or multiplied by the value of the last knot

`modifiers.apply.to.change`
If TRUE, after.modifier (the untransformed-by-link value) is multiplied by the change between the penultimate and last knots and added to the last knot to generate an additional knot at after.time (similarly, before.modifier is multiplied by the change from the first to the second and subtracted from the first). If FALSE, the transformed after.multiplier is added to the last knot after the last knot is transformed to modifier.link scale, and the resulting value is back-transformed to yield the knot at after.time (similarly, before.modifier is added to the first knot on modifier.link scale)

`modifier.link` The name of a transformation to the scale at which the before.modifer and after.modifier have alphas added

`modifier.min`, `modifier.max`
The min and max values for before.modifier and after.modifier

`overwrite.modifiers.with.alphas`
Logical indicating whether, when alphas are applied to the before.modifier or after.modifier, they OVERWRITE the modifiers. If F, alphas are added on the scale specified by modifier.link

`error.prefix` String to prepend to any error message in case of an error

---

`create.location.based.ifelse.likelihood.instructions`
Create Location Based If-Else Likelihood Instructions

Description

Create instructions that can instantiate a different 'jheem.likelihood' depending on the location.

Usage

```
create.location.based.ifelse.likelihood.instructions(
  ...,
  name = NULL,
  locations.list = list()
)
```

Arguments

...	'jheem.likelihood.instructions' objects that all have the same 'outcome.for.sim'.
name	Name for the ifelse; will default to
locations.list	A list of character vectors of locations that will use the corresponding instruction. The list should be length one shorter than the number of instructions. For example, if you have three instructions, the first vector has locations that will use the first instruction and second vector has locations that will use the second instruction. The third instruction will be used for all other locations.

---

create.log.linear.functional.form

Create a Functional Form that is Linear on the Log Scale

Description

Create a Functional Form that is Linear on the Log Scale

Usage

```
create.log.linear.functional.form(
  intercept,
  slope,
  anchor.year,
  min = 0,
  max = Inf,
  parameters.are.on.log.scale = T,
  overwrite.parameters.with.alphas = F,
  error.prefix = "Cannot create log-linear functional.form: "
)
```

Arguments

intercept, slope	Arrays or scalar values representing the intercept and slope on the log scale
anchor.year	The year at which the functional form evaluates to the intercept
min, max	The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)
parameters.are.on.log.scale	Logical indicating whether the intercept and slope are already on the log scale
overwrite.parameters.with.alphas	Logical indicating whether, when alphas are applied, they OVERWRITE values. If F, alphas are added to the transformed value (on the transformed scale)
error.prefix	String to prepend to any error message in case of an error

See Also

create.static.functional.form, create.linear.functional.form, create.logistic.linear.functional.form

---

create.logistic.linear.functional.form

Create a Functional Form that is Linear on the Logistic Scale

---

Description

Create a Functional Form that is Linear on the Logistic Scale

Usage

```
create.logistic.linear.functional.form(
  intercept,
  slope,
  anchor.year,
  min = 0,
  max = 1,
  parameters.are.on.logit.scale = T,
  overwrite.parameters.with.alphas = F,
  error.prefix = "Cannot create logistic-linear functional.form: "
)
```

Arguments

intercept, slope	Arrays or scalar values representing the intercept and slope on the logistic scale
anchor.year	The year at which the functional form evaluates to the intercept

`min`, `max` The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)

`parameters.are.on.logit.scale`
Logical indicating whether the intercept and slope are already on the logistic scale

`overwrite.parameters.with.alphas`
Logical indicating whether, when alphas are applied, they OVERWRITE values. If F, alphas are added to the transformed value (on the transformed scale)

`error.prefix` String to prepend to any error message in case of an error

See Also

create.static.functional.form, create.linear.functional.form, create.log.linear.functional.form

---

`create.logistic.spline.functional.form`
Create a Logistic Spline Functional Form

Description

Create a Logistic Spline Functional Form

Usage

```
create.logistic.spline.functional.form(
  knot.times,
  knot.values,
  min = -Inf,
  max = Inf,
  knot.link = c("identity", "log", "logit")[2],
  knot.min = NA,
  knot.max = NA,
  knots.are.on.transformed.scale = F,
  overwrite.knot.values.with.alphas = F,
  fraction.of.asymptote.after.end = 0.05,
  fraction.of.asymptote.before.start = 0.025,
  fraction.of.asymptote.for.change.dir = 0.02,
  fraction.asymptote.link = "log",
  overwrite.fraction.asymptote.with.alphas = F,
  error.prefix = "Cannot create logistic spline functional.form: "
)
```

Arguments

knot.times	The (numeric) times at which the knots of the spline apply. Must have names set
knot.values	Initial values for the knots. Must have names set, matching the names of knot times
min, max	The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)
knot.link	The name of a transformation to the scale at which the knot values have alphas ADDED. The knots are back-transformed from this scale prior to the spline scale being applied
knot.min, knot.max	The min and max values that KNOTS can take before being splined (note, the min and max values the functional form can take, after splining, are determined by parameters min and max)
knots.are.on.transformed.scale	Logical indicating whether the knots are already on the transformed scale (ie knot.link has been applied)
fraction.of.asymptote.after.end, fraction.of.asymptote.before.start	The fraction of the total change in the first or last segment that would take place before the first knot or after the last
fraction.of.asymptote.for.change.dir	The fraction of the total change in a segment that should take place before between two segments
fraction.asymptote.link	The name of a transformation to the scale at which alphas are added to the betas for the fraction.asymptote.<x> values
overwrite.fraction.asymptote.with.alphas	Logical indicating whether, when alphas are applied to the fraction.of.asymptote.<x>, they OVERWRITE the values. If F, alphas are interpreted as either additive or multiplicative to the knot.values, depending on the value of fraction.asymptote.alphas.are.multipliers
error.prefix	String to prepend to any error message in case of an error

---

create.logistic.tail.functional.form
Create a "Logistic Tail" Functional Form

Description

Create a "Logistic Tail" Functional Form

Usage

```
create.logistic.tail.functional.form(
  intercept,
  slope,
  anchor.year,
  logistic.after.frac.of.span = 0.5,
  min = 0,
  max = 1,
  intercept.link = "logistic",
  intercept.min = NA,
  intercept.max = NA,
  intercept.robust.to.negative = F,
  slope.link = intercept.link,
  slope.min = NA,
  slope.max = NA,
  slope.robust.to.negative = F,
  parameters.are.on.transformed.scale = F,
  overwrite.parameters.with.alphas = F,
  error.prefix = "Cannot create logistic-tail functional.form: "
)
```

Arguments

`intercept`, `slope`	Scalar values or arrays representing the intercept and slope. Note that the final form constrains the slope to be non-negative
`anchor.year`	The year at which the functional form evaluates to the intercept
`logistic.after.frac.of.span`	The fraction of the the total span (max - min) after which the model follows a logistic curve
`min`, `max`	The upper and lower limits to which the functional form can evaluate
`intercept.link`, `slope.link`	The names of the transformations to the scale on which alphas are added to calculate the intercept and slope. One of "identity", "log", "logit"
`intercept.min`, `intercept.max`, `slope.min`, `slope.max`	The min and max values to which intercept and slope can evaluate
`overwrite.parameters.with.alphas`	Logical indicating whether, when alphas are applied, they OVERWRITE values. If F, alphas are added to the transformed value (on the transformed scale)
`error.prefix`	String to prepend to any error message in case of an error

Details

A "logistic tail" functional form follows a linear (intercept + slope) in its first part, and a logistic-linear form in its second part. This allows a linear trend to taper off

`create.model.foreground` *Create a Model Foreground Object*

Description

Create a Model Foreground Object

Usage

```
create.model.foreground(..., error.prefix = "")
```

Arguments

`...`	The intervention.effects and target.populations for this foreground. Can be either: (1) an alternating set of jheem.intervention.effect objects and target.population objects (ie, one intervention.effect followed by one target.population, then the next pair, or vice versa) or (2) one list of intervention effects and one list of target populations, both of the same length

Details

A model foreground object applies multiple effects to multiple subpopulations *simultaneously*, with all effects targeting the same model quantity

`create.monotonic.criteria.based.intervention` *Create a "Guess-and-Check" Intervention that Must Satisfy Some Criteria by Varying Parameters*

Description

Create a "Guess-and-Check" Intervention that Must Satisfy Some Criteria by Varying Parameters

Usage

```
create.monotonic.criteria.based.intervention(
  base.intervention,
  completion.criteria,
  max.iterations,
  n.iterations.after.satisfying.criteria = 5,
  max.iterations.first.sim = max.iterations,
 n.iterations.after.satisfying.criteria.first.sim = ceiling(max.iterations.first.sim/2),
  max.failure.rate = 0,
  discount.prior.n = 0.5,
```

```
  code = NULL,
  name = NULL,
  overwrite.existing.intervention = F
)
```

Arguments

base.intervention	A single.iteration.intervention, as created by create.intervention or join.interventions
completion.criteria	Either a single jheem.intervention.criterion object, as created by create.intervention.criterion or a list containing only jheem.intervention.criterion objects
max.iterations	The maximum number of iterations to run in trying to satisfy intervention criteria. If criteria are not satisfied after max.iterations, will 'give up' (and either produce degenerate simulations or throw an error, depending on max.failure.rate)
n.iterations.after.satisfying.criteria	The number of iterations to run after we have satisfied the criteria to try to get as close as possible to the target
max.failure.rate	The maximum proportion of simulations in a simulation set which can fail to find a solution that satisfies intervention criteria within max.iterations. If this threshold is exceeded, will throw an error; otherwise, will return degenerate simulations when it could not find a solution
code	A brief (2-30 character) string that will uniquely identify this intervention

---

create.monotonic.criterion

Create a criterion for monotonic parameter, criteria-based interventions

---

Description

Create a criterion for monotonic parameter, criteria-based interventions

Usage

```
create.monotonic.criterion(
  parameter.name,
  outcome,
  parameter.scale,
  parameter.initial.value,
  min.acceptable.parameter.value = NULL,
  max.acceptable.parameter.value = NULL,
  target.value,
  min.acceptable.value,
  max.acceptable.value,
```

```
  subsequent.thresholds.apply.after.iteration = numeric(),
  dimension.values = list(),
  draw.parameter.from.previous.sims = !is.function(parameter.initial.value),
  ...
)
```

Arguments

`parameter.name`	The name of the parameter to be modified to achieve the outcome
`outcome`	A character vector indicating the name of the outcome to which the criterion applies
`parameter.scale`	The scale on which the parameter operates
`parameter.initial.value`	Either a single scalar numeric or a function that takes argument sim and returns a single scalar numeric
`min.acceptable.parameter.value`, `max.acceptable.parameter.value`	Indicators of the min and max possible values of the parameter to prevent unreasonably large (or small) parameter values crashing the algorithm. Must be single, numeric values or NULL, in which case the limits allowed by the 'parameter.scale' will be used.
`target.value`	A single, numeric value that we want the outcome to get to
`min.acceptable.value`, `max.acceptable.value`	Indicators of the min and max possible values of the outcome we would be willing to accept. Must be either (1) single, numeric values or (2) numeric vectors. In this case, the first values are the min/max threshold until subsequent.thresholds.apply.after.iteration[1], the second values are the thresholds prior to subsequent.thresholds.apply.after.iteration[2], etc
`subsequent.thresholds.apply.after.iteration`	The number of iterations after which the 2nd, 3rd, etc thresholds in min.acceptable.value, max.acceptable.value apply
`dimension.values`	The dimension values for which we should pull the outcome. Must be a named list of either character vectors, integer vectors, or logical vectors
`...`	An alternative way of supplying dimensions values - must be named character vectors, integer vectors, or logical vectors
`draw.parameters.from.previous.sims`	Whether the initial value for the parameter after the first simulation in a set should be the given initial value, or should be based on what worked for previous simulations in the set

Details

If the intervention algorithm finds it needs to keep raising (or lowering) the parameter value so high (or low) that it reaches the bounds, the parameter will be considered optimized. This often occurs when the parameter's outcome is already beyond the target and would be generally deemed acceptable anyway.

`create.natural.spline.functional.form` *Create a Natural Spline Functional Form*

Description

Create a Natural Spline Functional Form

Usage

```
create.natural.spline.functional.form(
  knot.times,
  knot.values,
  link = c("identity", "log", "logit")[1],
  min = NA,
  max = NA,
  knot.link = link,
  knot.min = min,
  knot.max = max,
  knots.are.on.transformed.scale = F,
  overwrite.knot.values.with.alphas = F,
  before.time = NULL,
  before.modifier = NULL,
  after.time = NULL,
  after.modifier = NULL,
  modifiers.apply.to.change = T,
  modifier.link = link,
  after.modifier.increasing.change.link = modifier.link,
  after.modifier.decreasing.change.link = modifier.link,
  before.modifier.increasing.change.link = modifier.link,
  before.modifier.decreasing.change.link = modifier.link,
  modifier.min = min,
  modifier.max = max,
  overwrite.modifiers.with.alphas = F,
  error.prefix = "Cannot create natural spline functional.form: "
)
```

Arguments

`knot.times`	The (numeric) times at which the knots of the spline apply. Must have names set
`knot.values`	Initial values for the knots. Must have names set, matching the names of knot times
`link`	The name of a transformation to the scale at which the SPLINE should apply. The knots are transformed to this scale, the spline is applied, and then the splined values are back-transformed. One of 'identity', 'log', or 'logistic'

`min`, `max` The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)

`knot.link` The name of a transformation to the scale at which the knot values have alphas ADDED. The knots are back-transformed from this scale prior to the spline scale being applied

`knot.min`, `knot.max`
The min and max values that KNOTS can take before being splined (note, the min and max values the functional form can take, after splining, are determined by parameters min and max)

`knots.are.on.transformed.scale`
Logical indicating whether the knots are already on the transformed scale (ie knot.link has been applied)

`before.time`, `after.time`
The times at which the knots calculated from before.modifier and after.modifier apply, respectively

`before.modifier`, `after.modifier`
If either of these is set, an additional knot is added to the specified knots. For before.modifier, this knot is calculated as before.modifier added or multiplier by the value of the first knot (on the model scale); after.modifier is added or multiplied by the value of the last knot

`modifiers.apply.to.change`
If TRUE, after.modifier (the untransformed-by-link value) is multiplied by the change between the penultimate and last knots and added to the last knot to generate an additional knot at after.time (similarly, before.modifier is multiplied by the change from the first to the second and subtracted from the first). If FALSE, the transformed after.multiplier is added to the last knot after the last knot is transformed to modifier.link scale, and the resulting value is back-transformed to yield the knot at after.time (similarly, before.modifier is added to the first knot on modifier.link scale)

`modifier.link` The name of a transformation to the scale at which the before.modifer and after.modifier have alphas added

`modifier.min`, `modifier.max`
The min and max values for before.modifier and after.modifier

`overwrite.modifiers.with.alphas`
Logical indicating whether, when alphas are applied to the before.modifier or after.modifier, they OVERWRITE the modifiers. If F, alphas are added on the scale specified by modifier.link

`error.prefix` String to prepend to any error message in case of an error

---

`create.nested.proportion.likelihood.instructions`
Create Nested Proportion Likelihood Instructions

Description

Create Nested Proportion Likelihood Instructions

Usage

```
create.nested.proportion.likelihood.instructions(
  outcome.for.data,
  denominator.outcome.for.data,
  outcome.for.sim,
  outcome.for.n.multipliers = denominator.outcome.for.data,
  location.types,
  minimum.geographic.resolution.type,
  maximum.locations.per.type = 3,
  dimensions = NULL,
  levels.of.stratification = 0:length(dimensions),
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
  exclude.denominator.ontology.names = NULL,
  location.overall.keep.threshold = 5,
  location.stratum.keep.threshold = 1,
  p.bias.inside.location,
  p.bias.outside.location,
  p.bias.sd.inside.location,
  p.bias.sd.outside.location,
  within.location.p.error.correlation = 0.5,
  within.location.n.error.correlation = 0.5,
  minimum.error.sd = 0,
  correlation.different.locations = 0,
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  obs.n.correlation.different.years = within.location.n.error.correlation,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  n.multiplier.cv = 0.1,
  p.error.variance.term = NULL,
  p.error.variance.type = NULL,
  n.error.variance.term = 0.05,
  n.error.variance.type = "cv",
  n.error.variance.term.if.estimated = 0.8,
  n.error.variance.type.if.estimated = "exp.of.variance",
  weights = list(),
  equalize.weight.by.year = T,
  partitioning.function,
  name = outcome.for.sim
)
```

Arguments

`outcome.for.data`
: A single character vector specifying the outcome that data managers will use to find data.

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`location.types`
: The types of the locations that contain or are contained by the model location. These should be in order of decreasing preference since in the case that two locations have the same contained 'minimum.geographic.resolution.type', such as county "11001" and state "DC", only the higher priority type location will be used.

`minimum.geographic.resolution.type`
: The type of location used to partition locations. The type of the model location AND 'location.types' types must all completely enclose regions of this type

`dimensions`
: A character vector of dimensions, excluding year, from which stratifications will be generated.

`levels.of.stratification`
: An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year`
: -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year`
: Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years`
: (Optional) An integer vector of years to ignore.

`sources.to.use`
: A character vector of sources from which to pull data.

`location.overall.keep.threshold`, `location.stratum.keep.threshold`
: How many data points a location must offer beyond what is found for the main location to justify being retained. Either overall (across ALL strata in total) or on a stratum-by-stratum basis. If a location doesn't meet the overall threshold, it will be ignored entirely. If it does, but fails to meet the stratum threshold for a certain stratum, it will be ignored only in that stratum.

`p.bias.inside.location`, `p.bias.outside.location`, `p.bias.sd.inside.location`, `p.bias.sd.outside.location`
: A single numeric value specifying the bias in the outcome proportion between locations inside (or outside) the model location and the model location itself, and their associated standard deviations. Each of these may alternatively be a function that takes only arguments 'version' and 'location' and returns an acceptable numeric value. For the estimates, values should be between -1 and 1, inclusive. For sd, should be non-negative and non-infinite. The bias should = outside/inside location MINUS main location (eg, state - MSA or county - MSA)

within.location.p.error.correlation, within.location.n.error.correlation
: Single numeric values specifying the correlation between p or n values from the same location and stratum

minimum.error.sd
: A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

correlation.different.locations
: A single numeric value specifying the correlation between observations of different locations.

correlation.different.years
: A single numeric value specifying the correlation between observations of different years.

correlation.different.strata
: A single numeric value specifying the correlation between observations from different strata.

correlation.different.sources
: A single numeric value specifying the correlation between observations from different sources.

correlation.same.source.different.details
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

obs.n.correlation.different.years
: Single numeric value specifying the correlation between different calculated obs n values

observation.correlation.form
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

n.multiplier.cv
: A single numeric value specifying the covariance term used to find the n.multipliers.

p.error.variance.type, p.error.variance.term
: The type of error variance(s) to be used for the outcome proportion and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

n.error.variance.type, n.error.variance.type
: The 'p.error.variance.type/term' analog for the denominator outcome. Has fewer options of type. See details for more information.

weights
: A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by create.likelihood.weights

equalize.weight.by.year
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

partitioning.function
: A function that partitions values in the data ontology into values in the model ontology. Must have two arguments, "arr" and "version" and return an array with the same dimnames as the input array.

Details

P error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with a 'p.error.variance.type' of "cv" and a 'p.error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like a 'p.error.variance.type' of 'c("data.sd", "function.sd") and a 'p.error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

N error variance works the same way but only one type is currently supported: "cv" (covariance).

See Also

Other Nested Proportion Likelihoods: `create.nested.proportion.likelihood.instructions.with.included.multip`
`create.time.lagged.comparison.nested.proportion.likelihood.instructions()`, `get.p.bias.estimates()`

---

create.nested.proportion.likelihood.instructions.with.included.multiplier

Create Nested Proportion Likelihood Instructions With Included Multiplier

---

Description

Create Nested Proportion Likelihood Instructions With Included Multiplier

Usage

```
create.nested.proportion.likelihood.instructions.with.included.multiplier(
  outcome.for.data,
  denominator.outcome.for.data,
  outcome.for.sim,
  outcome.for.n.multipliers = denominator.outcome.for.data,
  location.types,
  minimum.geographic.resolution.type,
  maximum.locations.per.type = 3,
  dimensions = NULL,
  levels.of.stratification = 0:length(dimensions),
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
```

```
  sources.to.use = NULL,
  exclude.denominator.ontology.names = NULL,
  location.overall.keep.threshold = 5,
  location.stratum.keep.threshold = 1,
  included.multiplier,
  included.multiplier.sd,
  included.multiplier.correlation = NULL,
  included.multiplier.correlation.structure = c("compound.symmetry",
    "autoregressive.1")[1],
  p.bias.inside.location,
  p.bias.outside.location,
  p.bias.sd.inside.location,
  p.bias.sd.outside.location,
  within.location.p.error.correlation = 0.5,
  within.location.n.error.correlation = 0.5,
  minimum.error.sd = 0,
  correlation.different.locations = 0,
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  obs.n.correlation.different.years = within.location.n.error.correlation,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  n.multiplier.cv = 0.1,
  p.error.variance.term = NULL,
  p.error.variance.type = NULL,
  n.error.variance.term = 0.05,
  n.error.variance.type = "cv",
  n.error.variance.term.if.estimated = 0.8,
  n.error.variance.type.if.estimated = "exp.of.variance",
  weights = list(),
  equalize.weight.by.year = T,
  partitioning.function,
  name = outcome.for.sim
)
```

Arguments

`outcome.for.data`
: A single character vector specifying the outcome that data managers will use to find data.

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`location.types`
: The types of the locations that contain or are contained by the model location. These should be in order of decreasing preference since in the case that two locations have the same contained 'minimum.geographic.resolution.type', such as county "11001" and state "DC", only the higher priority type location will be used.

`minimum.geographic.resolution.type`
: The type of location used to partition locations. The type of the model location AND 'location.types' types must all completely enclose regions of this type

`dimensions`
: A character vector of dimensions, excluding year, from which stratifications will be generated.

`levels.of.stratification`
: An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year`
: -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year`
: Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years`
: (Optional) An integer vector of years to ignore.

`sources.to.use`
: A character vector of sources from which to pull data.

`location.overall.keep.threshold`, `location.stratum.keep.threshold`
: How many data points a location must offer beyond what is found for the main location to justify being retained. Either overall (across ALL strata in total) or on a stratum-by-stratum basis. If a location doesn't meet the overall threshold, it will be ignored entirely. If it does, but fails to meet the stratum threshold for a certain stratum, it will be ignored only in that stratum.

`included.multiplier`
: The value of the multiplier that has already been used in calculating the simulation value of the outcome (represents the mean of the distribution around the multiplier).

`included.multiplier.sd`
: The standard deviation of the uncertainty around the included multiplier.

`included.multiplier.correlation`
: The correlation between values of the multiplier across different strata and years.

`included.multiplier.correlation.structure`
: Which correlation form to use for values of the multiplier across various strata and years.

`p.bias.inside.location`, `p.bias.outside.location`, `p.bias.sd.inside.location`, `p.bias.sd.outside.location`
: A single numeric value specifying the bias in the outcome proportion between locations inside (or outside) the model location and the model location itself, and their associated standard deviations. Each of these may alternatively be a function that takes only arguments 'version' and 'location' and returns an acceptable numeric value. For the estimates, values should be between -1 and 1, inclusive. For sd, should be non-negative and non-infinite. The bias should = outside/inside location MINUS main location (eg, state - MSA or county - MSA)

`within.location.p.error.correlation`, `within.location.n.error.correlation`
: Single numeric values specifying the correlation between p or n values from the same location and stratum

`minimum.error.sd`
: A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`correlation.different.locations`
: A single numeric value specifying the correlation between observations of different locations.

`correlation.different.years`
: A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
: A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
: A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`obs.n.correlation.different.years`
: Single numeric value specifying the correlation between different calculated obs n values

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`n.multiplier.cv`
: A single numeric value specifying the covariance term used to find the n.multipliers.

`p.error.variance.type`, `p.error.variance.term`
: The type of error variance(s) to be used for the outcome proportion and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`n.error.variance.type`
: The 'p.error.variance.type/term' analog for the denominator outcome. Has fewer options of type. See details for more information.

`weights`
: A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`partitioning.function`
: A function that partitions values in the data ontology into values in the model ontology. Must have two arguments, "arr" and "version" and return an array with the same dimnames as the input array.

Details

P error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with a 'p.error.variance.type' of "cv" and a 'p.error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like a 'p.error.variance.type' of 'c("data.sd", "function.sd") and a 'p.error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

N error variance works the same way but only one type is currently supported: "cv" (covariance).

See Also

Other Nested Proportion Likelihoods: `create.nested.proportion.likelihood.instructions()`, `create.time.lagged.comparison.nested.proportion.likelihood.instructions()`, `get.p.bias.estimates()`

`create.ontology.mapping` *Create a mapping that allows transformations of arrays*

Description

Create a mapping that allows transformations of arrays

Usage

```
create.ontology.mapping(
  mappings,
  from.dimensions,
  to.dimensions = from.dimensions,
  name = NULL,
  error.prefix = "Error creating ontology mapping: "
)
```

Arguments

`mappings`	A character matrix, with a number of columns == length(from.dimensions) + length(to.dimensions). The first length(from.dimensions) columns represent values for the from.dimensions (in the same order as from.dimensions), and the

last length(to.dimensions) colums represent values for to.dimensions (in the order of to.dimensions). Each row represents a mapping from a set of values for from.dimension to a set of values for to.dimensions. NB: any dimnames on mappings will be ignored

`from.dimensions`
: A character vector of names of the dimensions we are mapping from

`to.dimensions`
: A character vector of names of the dimensions we are mapping to

`error.prefix`
: A text string to prepend to any error messages generated in executing the function

---

`create.other.catchall.ontology.mapping`
Create an ontology mapping that lumps otherwise unspecified categories into an "other" category

---

Description

Create an ontology mapping that lumps otherwise unspecified categories into an "other" category

Usage

```
create.other.catchall.ontology.mapping(
  dimension,
  from.values,
  to.values,
  other.category.value = "other"
)
```

Arguments

`dimension`
: A single character value indicating what dimension this mapping applies to

`from.values`
: The possible values for the dimension in the ontology to map from

`to.values`
: The possible values for the dimension in the ontology to map to

`other.category.value`
: What the "other" category should be called in the to ontology

`create.outcome.location.mapping` *Create Outcome Location Mapping*

Description

Create Outcome Location Mapping

Usage

```
create.outcome.location.mapping(
  location.mappings,
  outcome.name,
  version = jheem.kernel$version,
  location = jheem.kernel$location,
  sub.version = NULL,
  jheem.kernel = NULL
)
```

Arguments

`location.mappings`
: is a named list, where the names correspond to modeled locations, and the elements are vectors of observed locations

`create.outcome.metadata` *Create a metadata object for tracked quantities or tracked transitions*

Usage

```
create.outcome.metadata(
  display.name,
  description,
  scale,
  axis.name,
  units,
  singular.unit = units,
  display.as.percent = scale == "proportion"
)
```

Arguments

display.name	The (nicely formatted) name of the quantity - to use in titling figures and other reporting. Should be capitalized
description	A phrase or single sentence describing the quantity. Should be captialized
scale	The scale of the quantity. Either "rate", "ratio", "proportion", "time", "number", "non.negative.number"
axis.name	A name to use in labeling axes on figures that display this quantity. Should be capitalized
units	The units the quantity takes (eg, "cases", "
	\itemsingular.unitThe unit to use if the value of the quantity is exactly == 1
	\itemdisplay.as.percentA logical indicator for whether, when plotted or otherwise displayed, the quantity should be converted to a percent. Only valid if scale==’proportion’

An object of class ’outcome.metadata’ which can be passed to track.transition, track.quantity, track.dynamic.quantity

Create a metadata object for tracked quantities or tracked transitions

---

create.simset.collection

Create a Collection of JHEEM Simulation-Sets

Description

Create a Collection of JHEEM Simulation-Sets

Usage

```
create.simset.collection(
  version,
  calibration.code,
  locations,
  interventions,
  n.sim = NULL,
  sub.version = NULL,
  root.dir = get.jheem.root.directory("Cannot create simset collection: ")
)
```

Arguments

version, sub.version	
	The version and (optional) sub-version for the simsets
locations	A character vector of locations for which to collect simsets
interventions	The interventions for which to collect interventions. Can be either (a) a character vector of intervention codes or (b) a list of ’jheem.intervention’ objects or single character intervention codes

n.sim	Optional - the number of simulations which collected simsets should contain. If NULL, the number will attempt to be figured out
root.dir	The root directory relative to which all simset files (and all JHEEM files) are saved

Value

An object of class 'jheem.simset.collection'

`create.solver.metadata`
Create Solver Metadata

Description

Create Solver Metadata

Usage

```
create.solver.metadata(
  method = "ode45",
  package = NULL,
  atol = 0.01,
  rtol = 0.001
)
```

`create.static.functional.form`
Create an Static (Time-Invariant) Functional Form

Description

Create an Static (Time-Invariant) Functional Form

Usage

```
create.static.functional.form(
  value,
  link = c("identity", "log", "logit")[1],
  min = NA,
  max = NA,
  value.name = "value",
  value.is.on.transformed.scale = F,
  overwrite.parameters.with.alphas = F,
  error.prefix = "Cannot create static functional.form: "
)
```

Arguments

value	array representing the value
link	The name of the transformation to the scale on which alphas are added to the value. One of "identity", "log", "logit"
min, max	The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)
value.is.on.transformed.scale	Logical indicating whether the value is already on the transformed scale at which alphas apply
overwrite.parameters.with.alphas	Logical indicating whether, when alphas are applied, they OVERWRITE values. If F, alphas are added to the transformed value (on the transformed scale)
error.prefix	String to prepend to any error message in case of an error

---

create.style.manager *Create A Style Manager*

Description

Create A Style Manager

Usage

```
create.style.manager(
  color.sim.by = "stratum",
  color.data.by = "stratum",
  shade.data.by = "location",
  linetype.sim.by = "simset",
  shape.data.by = "source",
  shape.sim.by = "simset",
  sim.palette = ggsci::pal_jama(),
  data.palette = ggsci::pal_jama(),
  linetypes = c("solid", "dashed", "dotted", "dotdash"),
  shapes = 21:25,
  shade.increment = -64,
  alpha.ribbon = 0.2,
  alpha.line = 1,
  linewidth.baseline = 1,
  linewidth.slope = 3
)
```

Arguments

`sim.palette, data.palette`
: Must be either a character vector of colors to choose from or a function that takes a number of colors needed and returns a character vector of colors.

`shapes`
: A numeric vector indicating which of the 25 ggplot shapes to select from. Each value must be an integer between 1 and 25.

`shade.increment`
: A single numeric value less than 0 indicating the extent of shading to apply (to alternate values of the 'shade.data.by'/'shade.sim.by' variables)

`alpha.line`
: A single numeric value which determines the transparency of the simulation lines. If NULL, the width of the lines is used (see "linewidth.slope" argument for details).

`linewidth.baseline`
: A single numeric value greater than zero which determines the simset linewidth in combination with the "linewidth.slope" argument.

`linewidth.slope`
: A single numeric value specifying how simset linewidth will be determined for based on how many simulations it has (its "n.sim"). The formula used is linewidth = baseline/(slope * log10(n.sim) + 1), where "baseline" is the argument "linewidth.baseline". If this slope is the default of 3, and linewidth.baseline is the default of 1, this will result in simsets with 1000 simulations having a linewidth of 0.1 while simsets with only 1 simulation will have a linewidth of 1. If you would like all simsets to plot with the same thickness, use a slope of 0.

`create.target.population` *Create a Target Population to Which Interventions May be Applied*

Description

Create a Target Population to Which Interventions May be Applied

Usage

```
create.target.population(..., name, invert = F)
```

Arguments

`...`
: Dimension values that denote the categories that comprise this target population. Each element must be namned with the name of the dimension, and can be either (1) character values, (2) integers, or (3) logical vectors

`name`
: A short (<= 20 characters) descriptive name for the target population. Should be nicely formatted for public display

`invert`
: A logical indicating whether the target population should be the INVERSE of the characteristics specified in ... (ie, everyone who does NOT fall into those categories)

See Also

Other Functions to create Target Populations: `diff.target.populations()`, `intersect.target.populations()`, `union.target.populations()`

`create.time.lagged.comparison.likelihood.instructions`
Create JHEEM Basic Likelihood Instructions

Description

Create JHEEM Basic Likelihood Instructions

Usage

```
create.time.lagged.comparison.likelihood.instructions(
  outcome.for.data,
  outcome.for.sim,
  dimensions = character(0),
  denominator.dimensions = dimensions,
  dimension.values = NULL,
  levels.of.stratification = NULL,
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
  minimum.error.sd = 0,
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  error.variance.term = NULL,
  error.variance.type = NULL,
  ratio.cv = NULL,
  ratio.correlation = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  use.lognormal.approximation = T,
  name = outcome.for.sim,
  na.rm = NULL
)
```

Arguments

`outcome.for.data`
: A single character vector specifying the outcome that data managers will use to find data.

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`dimensions`
: A character vector of dimensions, excluding year, from which stratifications will be generated.

`denominator.dimensions`
: (Optional) Analogous to 'dimensions' but for the simulation denominator outcome when applicable. Must be a subset of 'dimensions'.

`levels.of.stratification`
: An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year`
: -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year`
: Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years`
: (Optional) An integer vector of years to ignore.

`sources.to.use`
: A character vector of sources from which to pull data.

`minimum.error.sd`
: A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`correlation.different.years`
: A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
: A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
: A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`error.variance.type`, `error.variance.term`
: The type of error variance(s) to be used and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`weights`
: A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`use.lognormal.approximation`
: A single logical value.

`na.rm`
: Setting this TRUE will pass 'na.rm=T' along to the pull function. If set to NULL, 'na.rm=T' will be used for outcomes with scale 'proportion', 'rate', or 'ratio'.

Details

Error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with an 'error.variance.type' of "cv" and an 'error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like an 'error.variance.type' of 'c("data.sd", "function.sd") and an 'error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

See Also

Other Basic Likelihoods: `create.basic.likelihood.instructions()`, `create.basic.likelihood.instructions.wit`
`create.basic.likelihood.instructions.with.specified.outcome()`, `create.time.lagged.comparison.likeliho`

---

create.time.lagged.comparison.likelihood.instructions.with.included.multiplier

Create JHEEM Basic Likelihood Instructions

Description

Create JHEEM Basic Likelihood Instructions

Usage

```
create.time.lagged.comparison.likelihood.instructions.with.included.multiplier(
  outcome.for.data,
  outcome.for.sim,
  dimensions = character(0),
  denominator.dimensions = dimensions,
  dimension.values = NULL,
```

```
  levels.of.stratification = NULL,
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
  minimum.error.sd = 0,
  included.multiplier,
  included.multiplier.sd,
  included.multiplier.correlation = NULL,
  included.multiplier.correlation.structure = c("compound.symmetry",
    "autoregressive.1")[1],
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  error.variance.term = NULL,
  error.variance.type = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  use.lognormal.approximation = T,
  name = outcome.for.sim,
  na.rm = NULL
)
```

Arguments

`outcome.for.data`
: A single character vector specifying the outcome that data managers will use to find data.

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`dimensions`
: A character vector of dimensions, excluding year, from which stratifications will be generated.

`denominator.dimensions`
: (Optional) Analogous to 'dimensions' but for the simulation denominator outcome when applicable. Must be a subset of 'dimensions'.

`levels.of.stratification`
: An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year`
: -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year`
: Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years` (Optional) An integer vector of years to ignore.

`sources.to.use` A character vector of sources from which to pull data.

`minimum.error.sd`
: A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`included.multiplier`
: The value of the multiplier that has already been used in calculating the simulation value of the outcome (represents the mean of the distribution around the multiplier).

`included.multiplier.sd`
: The standard deviation of the uncertainty around the included multiplier.

`included.multiplier.correlation`
: The correlation between values of the multiplier across different strata and years.

`included.multiplier.correlation.structure`
: Which correlation form to use for values of the multiplier across various strata and years.

`correlation.different.years`
: A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
: A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
: A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`error.variance.type`, `error.variance.term`
: The type of error variance(s) to be used and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`weights` A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`use.lognormal.approximation`
: A single logical value.

`na.rm` Setting this TRUE will pass 'na.rm=T' along to the pull function. If set to NULL, 'na.rm=T' will be used for outcomes with scale 'proportion', 'rate', or 'ratio'.

Details

Error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with an 'error.variance.type' of "cv" and an 'error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like an 'error.variance.type' of 'c("data.sd", "function.sd") and an 'error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

See Also

Other Basic Likelihoods: `create.basic.likelihood.instructions()`, `create.basic.likelihood.instructions.wit`
`create.basic.likelihood.instructions.with.specified.outcome()`, `create.time.lagged.comparison.likeliho`

---

create.time.lagged.comparison.nested.proportion.likelihood.instructions

Create Time Lagged Comparison Nested Proportion Likelihood Instructions

---

Description

Create Time Lagged Comparison Nested Proportion Likelihood Instructions

Usage

```
create.time.lagged.comparison.nested.proportion.likelihood.instructions(
  outcome.for.data,
  denominator.outcome.for.data,
  outcome.for.sim,
  outcome.for.n.multipliers = denominator.outcome.for.data,
  location.types,
  minimum.geographic.resolution.type,
  maximum.locations.per.type = 3,
  dimensions = NULL,
  levels.of.stratification = 0:length(dimensions),
  from.year = -Inf,
  to.year = Inf,
  omit.years = NULL,
  sources.to.use = NULL,
```

```
  exclude.denominator.ontology.names = NULL,
  location.overall.keep.threshold = 5,
  location.stratum.keep.threshold = 1,
  p.bias.inside.location,
  p.bias.outside.location,
  p.bias.sd.inside.location,
  p.bias.sd.outside.location,
  within.location.p.error.correlation = 0.5,
  within.location.n.error.correlation = 0.5,
  minimum.error.sd = 0,
  correlation.different.locations = 0,
  correlation.different.years = 0.5,
  correlation.different.strata = 0.1,
  correlation.different.sources = 0.3,
  correlation.same.source.different.details = 0.3,
  obs.n.correlation.different.years = within.location.n.error.correlation,
  observation.correlation.form = c("compound.symmetry", "autoregressive.1")[1],
  n.multiplier.cv = 0.1,
  p.error.variance.term = NULL,
  p.error.variance.type = NULL,
  n.error.variance.term = 0.05,
  n.error.variance.type = "cv",
  n.error.variance.term.if.estimated = 0.8,
  n.error.variance.type.if.estimated = "exp.of.variance",
  ratio.cv = NULL,
  ratio.correlation = NULL,
  weights = list(),
  equalize.weight.by.year = T,
  partitioning.function,
  use.lognormal.approximation = T,
  name = outcome.for.sim
)
```

Arguments

`outcome.for.data`
: A single character vector specifying the outcome that data managers will use to find data.

`outcome.for.sim`
: A single character vector specifying the simulation outcome to use.

`location.types`
: The types of the locations that contain or are contained by the model location. These should be in order of decreasing preference since in the case that two locations have the same contained 'minimum.geographic.resolution.type', such as county "11001" and state "DC", only the higher priority type location will be used.

`minimum.geographic.resolution.type`
: The type of location used to partition locations. The type of the model location AND 'location.types' types must all completely enclose regions of this type

`dimensions` A character vector of dimensions, excluding year, from which stratifications will be generated.

`levels.of.stratification`
An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.

`from.year` -Inf or a single integer value specifying the earliest year for which data should be pulled.

`to.year` Inf or a single integer value specifying the latest year for which data should be pulled.

`omit.years` (Optional) An integer vector of years to ignore.

`sources.to.use` A character vector of sources from which to pull data.

`location.overall.keep.threshold`, `location.stratum.keep.threshold`
How many data points a location must offer beyond what is found for the main location to justify being retained. Either overall (across ALL strata in total) or on a stratum-by-stratum basis. If a location doesn't meet the overall threshold, it will be ignored entirely. If it does, but fails to meet the stratum threshold for a certain stratum, it will be ignored only in that stratum.

`p.bias.inside.location`, `p.bias.outside.location`, `p.bias.sd.inside.location`, `p.bias.sd.outside.location`
A single numeric value specifying the bias in the outcome proportion between locations inside (or outside) the model location and the model location itself, and their associated standard deviations. Each of these may alternatively be a function that takes only arguments 'version' and 'location' and returns an acceptable numeric value. For the estimates, values should be between -1 and 1, inclusive. For sd, should be non-negative and non-infinite. The bias should = outside/inside location MINUS main location (eg, state - MSA or county - MSA)

`within.location.p.error.correlation`, `within.location.n.error.correlation`
Single numeric values specifying the correlation between p or n values from the same location and stratum

`minimum.error.sd`
A single, positive numeric value. If any standard deviations are calculated to a value below this, they will be replaced with this value. This can help keep the likelihood from computing to negative infinity.

`correlation.different.locations`
A single numeric value specifying the correlation between observations of different locations.

`correlation.different.years`
A single numeric value specifying the correlation between observations of different years.

`correlation.different.strata`
A single numeric value specifying the correlation between observations from different strata.

`correlation.different.sources`
: A single numeric value specifying the correlation between observations from different sources.

`correlation.same.source.different.details`
: A single numeric value specifying the correlation between observations of the same source but which have different 'details'.

`obs.n.correlation.different.years`
: Single numeric value specifying the correlation between different calculated obs n values

`observation.correlation.form`
: Which correlation form to use in building the measurement error covariance matrix. May be 'compound.symmetry' or 'autoregressive.1'.

`n.multiplier.cv`
: A single numeric value specifying the covariance term used to find the n.multipliers.

`p.error.variance.type`, `p.error.variance.term`
: The type of error variance(s) to be used for the outcome proportion and the corresponding value or function required for each. See details for the allowable values. For more than one type, supply them in a vector. If you therefore need more than one term, supply these in a list.

`n.error.variance.type`
: The 'p.error.variance.type/term' analog for the denominator outcome. Has fewer options of type. See details for more information.

`weights`
: A list containing only numeric vectors and objects of class 'jheem.likelihood.weights', which are generated by `create.likelihood.weights`

`equalize.weight.by.year`
: A Boolean indicating whether every year should be weighted equally before the weights from 'weights' are applied.

`partitioning.function`
: A function that partitions values in the data ontology into values in the model ontology. Must have two arguments, "arr" and "version" and return an array with the same dimnames as the input array.

`use.lognormal.approximation`
: A single logical value.

Details

P error variance may be specified in a variety of ways, and more than one can be used in the same likelihood. 'error.variance.type' is a vector containing single character values describing the type(s) of variance to use, and there is one 'error.variance.term' corresponding to each. The types "sd" (standard deviation), "variance", "cv" (covariance), and "exp.of.variance" (exponent of variance) require a single numeric value as their term. The types "data.sd", "data.variance", and "data.cv" indicate that error data will be pulled from the data manager and matched directly to data points, and they require NULL as their term. Finally, the type "function.sd" requires as its term a function that takes only arguments "data", "details", "version", and "location" and returns a numeric array of the same dimensions as "data". Supplying multiple types (and therefore terms) results in the variances created by each being summed prior to use in the measurement error covariance matrix.

As an example, one type could be specified with a 'p.error.variance.type' of "cv" and a 'p.error.variance.term' of 0.13 (or list(0.13)). Two or more types might look like a 'p.error.variance.type' of 'c("data.sd", "function.sd") and a 'p.error.variance.term' of list(NULL, my_sd_function). It is possible to use the same type multiple times, although this is not recommended.

N error variance works the same way but only one type is currently supported: "cv" (covariance).

See Also

Other Nested Proportion Likelihoods: `create.nested.proportion.likelihood.instructions()`, `create.nested.proportion.likelihood.instructions.with.included.multiplier()`, `get.p.bias.estimates()`

---

declare.data.managers *Declare the data managers a specification uses*

Description

While a kernel for this specification's version is built under a binding, the binding must contain every declared name, and resolving any other name is an error. With no binding active, the declaration is not checked.

Usage

```
declare.data.managers(specification, names)
```

Arguments

`specification`	A jheem.specification object
`names`	The names of the data managers its functions use, for example c('census', 'syphilis')

Value

NULL, invisibly

---

diff.target.populations *Create a Target Population that is the DIFFERENCE between two other Target Populations*

Description

Create a Target Population that is the DIFFERENCE between two other Target Populations

Usage

```
## S3 method for class 'target.populations'
diff(pop1, pop2, name = NULL)
```

Arguments

pop1, pop2 Target Population Objects

Details

The resulting target population contains everyone who IS in pop1 but NOT in pop2

See Also

Other Functions to create Target Populations: `create.target.population()`, `intersect.target.populations()`, `union.target.populations()`

---

disable.jheem.solver.tracking

Disable JHEEM Solver Tracking

Description

Disable JHEEM Solver Tracking

Usage

```
disable.jheem.solver.tracking()
```

---

distribute.dimension.values

Distribute dimension values

Description

This function redistributes certain values in a dimension across the remaining values in that dimension. This is useful to remove problematic dimension values – like race = 'unknown' – while keeping the marginal sums and the relevant ratios between the remaining dimension values the same as they were before.

Usage

```
distribute.dimension.values(
  arr,
  dimension.values.to.distribute,
  round.distributed.dimension.values
)
```

Arguments

`arr`	An array – no validation is performed
`dimension.values.to.distribute`	A named list of the dimensions and their values which should be redistributed
`round.distributed.dimension.values`	Should distributed quantities be rounded?

Details

If multiple dimensions should be redistributed, they will be redistributed in the order that they exist in 'dimension.values.to.distribute'. This can result in different outputs for different orders.

Value

An array with the same number of dimensions as 'arr', but without the dimension values specified in 'dimension.values.to.distribute'.

`enable.jheem.solver.tracking` *Enable JHEEM Solver Tracking*

Description

Enable JHEEM Solver Tracking

Usage

```
enable.jheem.solver.tracking()
```

`examine.get.mappings.to.align.ontologies` *Identify any Reasons for a Failure to Get Mappings to Align Ontologies*

Description

For use in figuring out why a call to `get.mappings.to.align.ontologies` did not work

If no mappings can be found, prints detailed information on why we were unable to find mappings

Usage

```
examine.get.mappings.to.align.ontologies(
  ontology.1,
  ontology.2,
  align.on.dimensions = intersect(names(ontology.1), names(ontology.2)),
  allow.non.overlapping.incomplete.dimensions = F,
  include.dim.names = NULL,
  print.ontologies = T
)
```

Arguments

`ontology.1`, `ontology.2`
: The ontologies of two data elements to be transformed

`align.on.dimensions`
: The dimensions which should match in the transformed data

`allow.non.overlapping.incomplete.dimensions`
: Logical indicating whether, when we map to an incomplete dimension, there can be no overlap in the from and to values for that dimension

`include.dim.names`
: Optional argument, specifying specific dimension values that must be present in the transformed data

`print.ontologies`
: Whether to print the full ontologies in the details

`examine.get.ontology.mapping` *Identify any Reasons for a Failure to Map Ontologies*

Description

For use in figuring out why a call to `get.ontology.mapping` did not work

If no mapping can be found, prints detailed information on why we were unable to find a mapping

Usage

```
examine.get.ontology.mapping(
  from.ontology,
  to.ontology,
  allow.non.overlapping.incomplete.dimensions = F,
  print.ontologies = T
)
```

Arguments

from.ontology	The ontology to which the data to be transformed conform. Must be an 'ontology' object as created by ontology
to.ontology	The ontology to which the transformed data should conform. Must be an 'ontology' object as created by ontology
allow.non.overlapping.incomplete.dimensions	Logical indicating whether, when we map to an incomplete dimension, there can be no overlap in the from and to values for that dimension
print.ontologies	Whether to print the full ontologies in the details

---

examine.recent.failed.ontology.mapping

Get Detailed Information on a Recent Failed Attempt to Map Ontologies

---

Description

Prints detailed information on why we were unable to find a mapping

Usage

```
examine.recent.failed.ontology.mapping(which = 1, print.ontologies = T)
```

Arguments

which	The index of the recent failure. 1 denotes the most recent failed attempt, 2 the second most recent, and so on
print.ontologies	Whether to print the full ontologies in the details

---

execute.plotly.plot

Execute Plotly Plot A utility that performs the plot-rendering half of the plot simulations in plotly operation for applications like webtools that need the step to be separate.

Description

Execute Plotly Plot A utility that performs the plot-rendering half of the plot simulations in plotly operation for applications like webtools that need the step to be separate.

Usage

```
execute.plotly.plot(
  prepared.plot.data,
  outcomes = NULL,
  split.by = NULL,
  facet.by = NULL,
  plot.which = "sim.and.data",
  summary.type = c("individual.simulation", "mean.and.interval",
    "median.and.interval")[1],
  plot.year.lag.ratio = F,
  n.facet.rows = NULL,
  style.manager = get.default.style.manager(),
  debug = F,
  hide.legend = FALSE
)
```

---

execute.simplot *Execute Simplot A utility that performs the plot-rendering half of the simplot operation for applications like webtools that need the step to be separate.*

Description

Execute Simplot A utility that performs the plot-rendering half of the simplot operation for applications like webtools that need the step to be separate.

Usage

```
execute.simplot(
  prepared.plot.data,
  outcomes = NULL,
  split.by = NULL,
  facet.by = NULL,
  plot.which = c("sim.and.data", "sim.only", "data.only")[1],
  summary.type = c("individual.simulation", "mean.and.interval",
    "median.and.interval")[1],
  plot.year.lag.ratio = F,
  n.facet.rows = NULL,
  style.manager = get.default.style.manager(),
  debug = F
)
```

Arguments

`prepared.plot.data`
: The named list output by "prepare.plot"

outcomes	A character vector of which simulation outcomes to plot
split.by	At most one dimension
facet.by	Any number of dimensions but cannot include the split.by dimension
plot.which	Should simulation data and calibration data be plotted ('sim.and.data'), or only simulation data ('sim.only')
n.facet.rows	An optional argument to specify how many rows the figure should facet plots into
style.manager	We are going to have to define this down the road. It's going to govern how we do lines and sizes and colors. For now, just hard code those in, and we'll circle back to it

---

`expand.array` *Expand an Array to Greater Dimensions*

Description

Expand an Array to Greater Dimensions

Usage

```
expand.array(to.expand, target.dim.names)
```

Arguments

to.expand	The value to be expanded. Can be either (1) an array or matrix with named dimensions and dimnames set, (2) a scalar value, or (3) a named vector whose names are a superset of one of the elements of target.dim.names
target.dim.names	A named list containing the values of dimensions to subset for. The names of dimension.values should correspond to the names of dimnames. The elements of dimension.values should be character, integer, or logical vectors

---

`extract.last.simulation.from.calibration`
Extract the Last Simulation From A Calibration

Description

Extract the Last Simulation From A Calibration

Usage

```
extract.last.simulation.from.calibration(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory("Cannot set up calibration: "),
  allow.incomplete = T,
  include.first.sim = F,
  chains = NULL
)
```

Arguments

version	The version of the model specification (must have been previously registered)
location	A single character value representing the location for the calibration
include.first.sim	Whether to also pull the first simulation from the calibration (will then return a 2-simulation simset)
chains	

---

find.ontology.mapping *Find Ontology Mapping*

Description

Find Ontology Mapping

Usage

```
find.ontology.mapping(
  from.ontology,
  to.ontology,
  required.dimensions,
  required.dim.names,
  get.two.way.alignment,
  allow.non.overlapping.incomplete.dimensions,
 from.ontology.name = ifelse(get.two.way.alignment, "ontology.1", "the from.ontology"),
 to.ontology.name = ifelse(get.two.way.alignment, "ontology.2", "the to.ontology"),
  get.details = F
)
```

Arguments

from.ontology, to.ontology	The ontologies of the from and to arrays

required.dimensions
: The dimensions that should be present in the aligned product

required.dim.names
: Any specific dimension values that should be present in the aligned product

get.two.way.alignment
: Logical indicating whether we need one mapping, that maps from.dim.names such that

allow.non.overlapping.incomplete.dimensions
: Logical indicating whether, when we map to an incomplete dimension, there can be no overlap in the from and to values for that dimension

Value

If get.two.way.alignment is false, returns a list of one or more ontology.mapping objects, such that applying these mappings to an array with from.dim.names ('from.arr') would yield an array with dimnames ('to.dim.names')

If get.two.way.alignment is true, returns a list with two elements, each of which is itself a list of one or more ontology.mapping objects, such that applying the first set of mappings to an array with from.dimnames and applying the second set of mappings to an array with to.dimnames would both yield arrays with dimensions 'align.on.dimensions', both with the same dimnames

---

get.calibration.progress
Get Calibration Progress

Description

Summarize the progress of a calibration task

Usage

```
get.calibration.progress(
  version,
  locations,
  calibration.code,
  root.dir = get.jheem.root.directory(),
  round.to.digits = ifelse(as.pct, 0, 2),
  as.pct = T
)
```

Arguments

version, calibration.code, locations
: The version, locations, and calibration code for which to gauge progress

root.dir
: The root directory from which to search for calibrations

round.to.digits
: How many digits to round progress to

as.pct
: Whether the progress should be shown as a percentage instead of a fraction

---

`get.collapse.array.indices` *Get a Set of Indices to Collapse One Array into Another*

Description

Get a Set of Indices to Collapse One Array into Another

Usage

```
get.collapse.array.indices(large.arr.dim.names, small.arr.dim.names)
```

Arguments

`large.arr.dim.names`	The dimnames of the source array
`small.arr.dim.names`	the dimnames of the target array

Value

A list with four elements: $large.indices, $small.indices, $large.n, $small.n large.indices and small.indices are integer vectors of equal length, such that the value at large.indices[i] goes into the value at small.indices[i] if collapsing

---

`get.cumulative.age.counts.smoother` *Get a function that projects a smoothed cumulative count up to a given age*

Description

Get a function that produces a cubic spline over the *cumulative* counts by age up to a given age

Usage

```
get.cumulative.age.counts.smoother(
  counts,
  given.age.brackets,
  smooth.infinite.age.to = Inf,
  allow.extrapolation = F,
  method = c("monoH.FC", "hyman")[1],
  error.prefix = ""
)
```

Arguments

allow.extrapolation

---

get.data.binding *Get the active data binding*

Description

Get the active data binding

Usage

```
get.data.binding()
```

Value

The binding made active most recently by using.data.binding or a jheem2 function building under it; otherwise the session binding if set.data.binding was called; otherwise NULL

---

get.data.binding.identity *Get the identity of a data binding*

Description

Get the identity of a data binding

Usage

```
get.data.binding.identity(binding)
```

Arguments

binding	A 'jheem.data.binding' object

Value

A data frame with one row per name: name, data.manager.name, and the file, release, repository, sha256, md5, and modified status of the data manager bound to it (NA where unknown; see get.data.manager.provenance)

`get.data.inputs` *Get the data inputs recorded for an artifact*

Description

The data managers that were in use when a kernel, likelihood, simulation set, or calibration was built: those registered by name, the default data manager, a manager passed in explicitly, and every manager loaded with `load.data.manager` in the session. The record can include managers the artifact did not use.

Usage

```
get.data.inputs(x)
```

Arguments

`x`	A jheem.kernel, jheem.likelihood, or jheem.simulation.set, or the path of a calibration directory or of the data_inputs.dcf file in one. If missing, the record an artifact built now would get (under a data binding, with no names used yet).

Value

NULL if nothing was recorded (for example, for artifacts built by earlier versions of jheem2). Otherwise a list with

coverage 'declared' if any managers were registered by name, 'loaded', 'bound' if the artifact was built under a data binding, and 'unknown' if any listed manager has no identity (it was not loaded from a file)

recorded.at When the record was made

run.record The JHEEM_RUN_RECORD environment variable at that time, if it was set: a pointer to the run's own record, for example a container run's records directory

data.managers A data frame with one row per data manager: registered.as (the registered names, NA if none), default, passed, data.manager.name, file, release, repository, sha256, md5, modified (whether its contents had changed since loading; NA when not known) and loaded.at; and, for an artifact built under a data binding, bound.as (the names it was bound under, NA if none) and used (whether the build resolved any of those names; NA if not bound)

get.data.manager *Get a data manager by name*

Description

With a data binding active (see get.data.binding), the data manager bound to the name, and the name is recorded as used by any model object being built. Otherwise, the data manager registered under the name.

Usage

```
get.data.manager(name)
```

Arguments

name	The name it was bound or registered under

Value

A jheem.data.manager

get.data.manager.provenance *Get the provenance of a data manager*

Description

Where a data manager came from: the file it was loaded from, an MD5 digest of that file, the release it belongs to when the loader supplied it, when it was loaded, and whether its contents have changed since.

Usage

```
get.data.manager.provenance(data.manager)
```

Arguments

data.manager	A jheem.data.manager object

Value

A list with elements file, md5, sha256, release, repository, loaded.at and modified, or NULL for a data manager that was not loaded from a file (or was loaded by an earlier version of jheem2)

`get.data.outcome.pretty.names` *Get pretty names for outcomes*

Description

Get pretty names for outcomes

Usage

```
get.data.outcome.pretty.names(
  data.manager = get.default.data.manager(),
  outcomes
)

get.data.outcome.labels(data.manager = get.default.data.manager(), outcomes)

get.data.outcome.descriptions(
  data.manager = get.default.data.manager(),
  outcomes
)
```

Arguments

`data.manager`	A jheem.data.manager object
`outcomes`	A character vector with the names of outcomes previously registered to this data manager

Details

Gets the pretty.names, labels, or descriptions registered for the outcomes with the data manager

Value

A character vector of pretty names, labels, or descriptions

Functions

- `get.data.outcome.labels()`: Get the labels for outcomes
- `get.data.outcome.descriptions()`: Get descriptions for outcomes

`get.data.source.full.names` *Get full names for data sources*

Description

Get full names for data sources

Usage

```
get.data.source.full.names(data.manager = get.default.data.manager(), sources)

get.data.source.short.names(data.manager = get.default.data.manager(), sources)
```

Arguments

`data.manager`	A jheem.data.manager object
`sources`	A character vector with the names of sources previously registered to this data manager

Details

Gets the full.names or short.names for the data sources in the data manager

Value

A character vector of full names or short names

Functions

- `get.data.source.short.names()`: Get short names for data sources

`get.default.data.manager` *Get the default data manager*

Description

Get the default data manager

Usage

```
get.default.data.manager()
```

`get.default.style.manager`
Get the Default Style Manager

Description

Get the Default Style Manager

Usage

```
get.default.style.manager(type = c("ggplot", "plotly")[1])
```

`get.expand.array.indices`
Get the Array Indices Under Expanded Dim-names

Description

Get the Array Indices Under Expanded Dim-names

Usage

```
get.expand.array.indices(to.expand.dim.names, target.dim.names, index.from = 1)
```

`get.expand.then.collapse.array.indices`
Get a Set of Indices to Collapse One Array into Another, expanding the first array if needed to add dimensions

---

Description

Get a Set of Indices to Collapse One Array into Another, expanding the first array if needed to add dimensions

Usage

```
get.expand.then.collapse.array.indices(
  large.arr.dim.names,
  small.arr.dim.names
)
```

Arguments

large.arr.dim.names
: The dimnames of the source array

small.arr.dim.names
: the dimnames of the target array

Value

A list with four elements: $large.indices, $small.indices, $large.n, $small.n large.indices and small.indices are integer vectors of equal length, such that the value at large.indices[i] goes into the value at small.indices[i] if collapsing

---

get.identity.ontology.mapping

Get an "Identity" Ontology Mapping (that does not change data structure when applied)

Description

Get an "Identity" Ontology Mapping (that does not change data structure when applied)

Usage

```
get.identity.ontology.mapping()
```

Value

An object of class "ontology.mapping"

See Also

Other Functions to identify why ontologies fail to map: `apply.ontology.mapping()`, `map.value.ontology()`

---

get.jheem.root.directory

Get JHEEM Root Directory #' @description Gets the directory that calibration directories and files will be written into, if it has been set with "set.jheem.root.directory"

Description

Get JHEEM Root Directory #' @description Gets the directory that calibration directories and files will be written into, if it has been set with "set.jheem.root.directory"

Usage

```
get.jheem.root.directory(error.prefix = "")
```

`get.jheem.solver.tracked.info` *Get JHEEM Solver Tracked Info*

Description

Get JHEEM Solver Tracked Info

Usage

```
get.jheem.solver.tracked.info()
```

`get.locations.with.data` *Get locations with data for an outcome*

Description

Get locations with data for an outcome

Usage

```
get.locations.with.data(
  data.manager = get.default.data.manager(),
  outcome,
  metric = "estimate",
  years,
  exclude.ontology.names = NULL
)
```

Arguments

`outcome`	The outcome for which to find earliest and latest years of data (must be registered in the data manager)
`metric`	The type of measurement to find earliest and latest years of data within the outcome. The default value is "estimate", but other options include "cv", "variance", and "sd".
`years`	The years to include. May be NULL to search all available years, or a numeric vector with no NA's or duplicates.
`exclude.ontology.names`	The names of ontologies to exclude from this search. Must be NULL or a character vector with no NA's or empty values.

Details

Get locations that have data for a given outcome, metric and set of years. Searches all sources and all ontologies that haven't been specifcally excluded

get.mappings.to.align.ontologies

Get a Pair of Ontology Mappings that Aligns two Data Elements

Description

Get a Pair of Ontology Mappings that Aligns two Data Elements

Usage

```
get.mappings.to.align.ontologies(
  ontology.1,
  ontology.2,
  align.on.dimensions = intersect(names(ontology.1), names(ontology.2)),
  allow.non.overlapping.incomplete.dimensions = F,
  include.dim.names = NULL
)
```

Arguments

`ontology.1`, `ontology.2`
: The ontologies of two data elements to be transformed

`align.on.dimensions`
: The dimensions which should match in the transformed data

`allow.non.overlapping.incomplete.dimensions`
: Logical indicating whether, when we map to an incomplete dimension, there can be no overlap in the from and to values for that dimension

`include.dim.names`
: Optional argument, specifying specific dimension values that must be present in the transformed data

Value

Either a (1) NULL - if no mappings could align the ontologies or (2) a list with two elements, each an ontology.mapping object, such that applying the first to data conforming to ontology.1 and applying the second to data conforming to ontology.2 yields two data objects with the same dim.names

get.mcmc.mixing.statistic

Get MCMC Mixing Statistic

Description

For one chain, the mixing statistic is the variance in the first quarter of simulations divided by the variance in the last quarter of simulations.

Usage

```
get.mcmc.mixing.statistic(
  simset,
  match.names = NULL,
  chains = simset$unique.chains,
  sort = T
)
```

Arguments

match.names	(Optional) A regex on parameter names for which to compute the mixing statistic. If NULL, all parameters are used.
chains	(Optional) Which chains to use, defaulting to all chains.
sort	(Optional) Whether to sort the output in decreasing order by mixing statistic value, defaulting to TRUE.

---

get.ontology.error.debug.info

Get Ontology Error Debug Info

Description

Returns information useful for debugging ontology mapping errors

Usage

```
get.ontology.error.debug.info()
```

---

get.ontology.mapping

Get an Ontology Mapping to Transform Data

Description

Get an Ontology Mapping to Transform Data

Usage

```
get.ontology.mapping(
  from.ontology,
  to.ontology,
  allow.non.overlapping.incomplete.dimensions = F
)
```

Arguments

from.ontology	The ontology to which the data to be transformed conform. Must be an 'ontology' object as created by ontology
to.ontology	The ontology to which the transformed data should conform. Must be an 'ontology' object as created by ontology
allow.non.overlapping.incomplete.dimensions	Logical indicating whether, when we map to an incomplete dimension, there can be no overlap in the from and to values for that dimension

Value

Either an object of class 'ontology.mapping' or NULL if no mapping that would bridge the differences is found

---

get.ontology.mapping.matrix

Get a Matrix that Applies an Ontology Mapping

Description

Get a Matrix that Applies an Ontology Mapping

Usage

```
get.ontology.mapping.matrix(
  ontology.mapping,
  from.dim.names,
  to.dim.names = NULL,
  error.prefix = ""
)
```

Arguments

ontology.mapping	An object of class 'ontology.mapping' - obtained either through register.ontology.mapping, get.ontology.mapping, get.mappings.to.align.ontologies
to.dim.names	The dimnames that the resulting transformed data should have (can represent a subset of the dimnames that the ontology mapping knows how to transform). If NULL, the ontology mapping will keep the dimnames in from.arr except for dimensions that are transformed
error.prefix	A character value to be prepended to any error messages thrown by the function

`get.p.bias.estimates` *Get P Bias Estimates*

Description

Get P Bias Estimates

Usage

```
get.p.bias.estimates(
  data.manager = get.default.data.manager(),
  dimensions,
  levels.of.stratification,
  outcome.for.p,
  outcome.for.n,
  sub.location.type,
  super.location.type,
  main.location.type = "CBSA",
  minimum.sample.size = 12,
  from.year = NULL,
  to.year = NULL,
  main.location.type.p.source = NULL,
  sub.location.type.p.source = NULL,
  super.location.type.p.source = NULL,
  main.location.type.n.source = NULL,
  sub.location.type.n.source = NULL,
  super.location.type.n.source = NULL,
  debug = F
)
```

Arguments

`data.manager`	The name of the data manager
`dimensions`	A character vector of dimensions, excluding year, from which stratifications will be generated.
`levels.of.stratification`	An integer vector specifying how the dimensions should be combined to form strata. '0' indicates totals (not stratified) while '1' indicates strata that are each stratified by one dimension at a time, '2' indicates strat that are each stratified by a combination of two dimensions at a time, etc. May not exceed the number of dimensions. Defaults to NULL, which is equivalent to '0'.
`outcome.for.p, outcome.for.n`	Outcome names for proportion and denominator data, respectively, as found in the data manager
`sub.location.type`	Used to find p-bias "inside" estimates by comparing data for this type against that of the main location type. If NULL, no p-bias "inside" estimates are found.

`super.location.type`
: Used to find p-bias "outside" estimates by comparing data for this type against that of the main location type. If NULL, no p-bias "outside" estimates are found.

`main.location.type`
: The location type that serves as the basis of comparison.

`minimum.sample.size`
: Numeric value specifying the minimum number of sub-main location pairs (or super-main location pairs) that must be found in order to calculate p-bias estimates

`from.year`, `to.year`
: Both NULL or both single numeric values

Value

A list containing mean and sd for p-bias "inside" and p-bias "outside" estimates. Some or all of these may be NA if 'sub.location.type' and/or 'super.location.type' were left NULL.

See Also

Other Nested Proportion Likelihoods: `create.nested.proportion.likelihood.instructions()`, `create.nested.proportion.likelihood.instructions.with.included.multiplier()`, `create.time.lagged.comp`

`get.prior.versions` *Get Prior JHEEM Model Versions*

Description

Get Prior JHEEM Model Versions

Usage

```
get.prior.versions(version, recursive = T)
```

See Also

Other JHEEM Version Management Functions: `get.specification.for.version()`, `register.model.specification()`, `version.has.been.registered()`

get.registered.ontology *param ontology.name The name of an ontology (must be registered in the data manager)*

Description

param ontology.name The name of an ontology (must be registered in the data manager)

Usage

```
get.registered.ontology(
  data.manager = get.default.data.manager(),
  ontology.name
)
```

get.simset.data *Get Simulation Data*

Description

Get Simulation Data

Usage

```
get.simset.data(
  simset,
  outcomes,
  output = c("value", "numerator", "denominator")[[1]],
  keep.dimensions = NULL,
  dimension.values = list(),
  ...,
  check.consistency = T,
  drop.single.outcome.dimension = T,
  error.prefix = "Error getting simulation results: "
)
```

Arguments

outcomes	A character vector with one or more outcomes for which to pull data. Must be a subset of sim$outcomes
keep.dimensions	Which dimensions should be retained in the resulting array
dimension.values	A set of dimension values according to which to subset the data (will be combined with ...)

`...` The parameters in ... are interpreted as additional dimension values according to which to subset the data. Each must be named, and either a character, integer, or logical vector

`check.consistency`
Whether to check the consistency of arguments. Setting to FALSE will yield faster performance but unintelligible error messages

`error.prefix` A character value to prepend to any errors generating in getting these data

`drop.single.outcome.dimensions`
When set to FALSE, the resulting array will have 'outcome' as its last dimension regardless of how many outcomes we are getting data for. If TRUE, the returned array will omit that dimension when only one outcome is requested

Details

Returns an array whose dimensions are keep.dimensions - plus (if there is more than one outcome or drop.single.outcome.dimensions==FALSE) an 'outcome' dimension at the end

---

`get.simset.filename` *Get Simset Filename*

Description

Get Simset Filename

Usage

```
get.simset.filename(
  version,
  location,
  calibration.code,
  n.sim,
  intervention.code = NULL,
  sub.version = NULL,
  include.path = T,
  root.dir = get.jheem.root.directory("Cannot get.simset.filename(): "),
  error.prefix = ""
)
```

get.simset.filename.and.dir *Get Simset Filename and Directory*

Description

Get Simset Filename and Directory

Usage

```
get.simset.filename.and.dir(
  version,
  location,
  calibration.code,
  n.sim,
  intervention.code = NULL,
  sub.version = NULL,
  root.dir = get.jheem.root.directory("Cannot get.simset.filename(): "),
  error.prefix = ""
)
```

get.simset.params *Get Simulation Set Paramater Values*

Description

Get Simulation Set Paramater Values

Usage

```
get.simset.params(
  simset,
  match.names = NULL,
  simulation.indices = simset$n.sim,
  drop = T
)
```

Arguments

match.names	A regex to select parameters by name. If NULL, all parameters are used.
simulation.indices	An integer or logical vector specifying which simulations to show in output. Defaults to the last simulation.
drop	If the default TRUE, then an output with only one simulation will be coerced to a vector.

`get.simulation.metadata` *Get a Simulation Metadata Object*

Description

Get a Simulation Metadata Object

Usage

```
get.simulation.metadata(
  version,
  location,
  from.year = NULL,
  to.year = NULL,
  n.sim = 1,
  sub.version = NULL,
  jheem.kernel = NULL,
  update.labels = T,
  error.prefix = paste0("Error deriving the simulation-metadata for '", version,
    "' and location '", location, "': ")
)
```

Arguments

`version`	The version of the model specification (must have been previously registered) for which to get metadata
`location`	A single character value representing the location for the metadata
`from.year`, `to.year`	The years which a corresponding simulation will have data for
`jheem.kernel`	Optional: the jheem.kernel on which to base the simulation metadata
`update.labels`	If jheem.kernel is not NULL, whether to get updated labels from the latest specification (vs the specification as it was when the kernel was created)
`error.prefix`	A string to prepend to any errors generated in getting the metadata object

Details

A simulation.metadata object contains metadata, such as the dim.names to which its contents will conform

---

`get.specification.for.version`
Get a JHEEM Specification Object

Description

Get a JHEEM Specification Object

Usage

```
get.specification.for.version(version)
```

Arguments

`version`	The name of the version under which the specification was registered

Value

An object of class 'jheem.specification'

See Also

Other JHEEM Version Management Functions: `get.prior.versions()`, `register.model.specification()`, `version.has.been.registered()`

`get.specification.metadata`
Get a Specification Metadata Object

Description

Get a Specification Metadata Object

Usage

```
get.specification.metadata(version, location, error.prefix = "")
```

Arguments

`version`	The version of the model specification (must have been previously registered) for which to get metadata
`location`	A single character value representing the location for the metadata
`error.prefix`	A string to prepend to any errors generated in getting the metadata object

Details

A specification.metadata object contains metadata, such as dim.names of quantities, for a specification

get.year.bounds.for.outcome *Get year bounds for outcome*

Description

Get year bounds for outcome

Usage

```
get.year.bounds.for.outcome(
  data.manager = get.default.data.manager(),
  outcome,
  metric = "estimate",
  exclude.ontology.names = NULL
)
```

Arguments

outcome	The outcome for which to find earliest and latest years of data (must be registered in the data manager)
metric	The type of measurement to find earliest and latest years of data within the outcome. The default value is "estimate", but other options include "cv", "variance", and "sd".
exclude.ontology.names	The names of ontologies to exclude from this search. Must be NULL or a character vector with no NA's or empty values.

Details

Gets the earliest and latest years for which a data manager has data for a given outcome and metric. Searches all sources and all ontologies that haven't been specifically excluded

import.data *Import data from another JHEEM Data Manager*

Description

Import data from another JHEEM Data Manager

Usage

```
import.data(to.data.manager = get.default.data.manager(), from.data.manager)
```

Arguments

`to.data.manager`
: A jheem.data.manager object

`from.data.manager`
: A jheem.data.manager object

`incomplete.dimensions` *Get an ontology's incomplete dimensions*

Description

Get an ontology's incomplete dimensions

Usage

```
incomplete.dimensions(x)
```

Arguments

`ontology`
: An ontology, as created by `ontology`

Value

A vector containing the names of the ontology's incomplete dimensions

---

`incomplete.dimensions.ontology` *Get an ontology's incomplete dimensions*

Description

Get an ontology's incomplete dimensions

Usage

```
## S3 method for class 'ontology'
incomplete.dimensions(x)
```

Arguments

`ontology`
: An ontology, as created by `ontology`

Value

A vector containing the names of the ontology's incomplete dimensions

`inspect.marginals` *Inspect marginal sums between stratifications*

Description

Compares the percentage difference between the marginal sums of more stratified data and the data it corresponds to in more aggregated stratifications of the same source and ontology.

Usage

```
inspect.marginals(
  data.manager = get.default.data.manager,
  outcome = NULL,
  source = NULL,
  ontology = NULL,
  sub.stratification = NULL,
  super.stratification = NULL,
  years = NULL,
  locations = NULL,
  threshold.percent = 10,
  threshold.magnitude = 50,
  check.up.to.n.way.stratified = 1
)
```

Arguments

`data.manager`	A jheem.data.manager object
`outcome`	The outcome type for the data. Must be an outcome previously registered with `register.data.outcome`
`source`	The (single) character name of the source from which these data derive. Must be a source previously registered with `register.data.source` Note: a source is conceived such that one source cannot contain two sets of data for the same set of dimension values
`ontology`	The name of a registered ontology
`sub.stratification`	The name of a less-aggregated stratification
`super.stratification`	The name of a more-aggregated stratification
`years, locations`	Show results for certain years (numeric, non-NA vector) or locations (character, non-NA vector)
`threshold.percent`	A percentage below which discrepancies will not be reported (default 10).
`threshold.magnitude`	A magnitude of difference between marginal sums and more-aggregated data below which discrepancies will not be reported (default 50).

check.up.to.n.way.stratified
: An integer greater than 0 indicating the highest level of stratification to report marginal sum discrepancies for (default 1).

`instantiate.likelihood` *Instantiate Likelihood*

Description

Instantiate Likelihood

Usage

```
instantiate.likelihood(
  instructions,
  version,
  location,
  sub.version = NULL,
  data.manager = get.default.data.manager(),
  verbose = F,
  throw.error.if.no.data = F,
  error.prefix = "Error instantiating likelihood from instructions: "
)
```

Arguments

instructions	A 'jheem.likelihood.instructions' object
data.manager	A jheem.data.manager object, or the name of one in the active data binding (see get.data.manager). May be NULL for Bernoulli likelihood instructions.
verbose	Should detailed messages be printed out as we go
throw.error.if.no.data	Should an error be thrown if no relevant data is found?

`interpolate` *Interpolate Values*

Description

Interpolate Values

Usage

```
interpolate(values, value.times, desired.times)
```

intersect.target.populations

Create a Target Population that is the INTERSECTION of Other Target Populations

Description

Create a Target Population that is the INTERSECTION of Other Target Populations

Usage

```
intersect.target.populations(..., name = NULL)
```

Arguments

...	One or more target.population objects or lists of target populations
name	A short (<= 20 characters) descriptive name for the target population. Should be nicely formatted for public display

See Also

Other Functions to create Target Populations: `create.target.population()`, `diff.target.populations()`, `union.target.populations()`

is.ontology

Test if an object is of type 'ontology'

Description

Test if an object is of type 'ontology'

Usage

```
is.ontology(x)
```

Arguments

x	Object to be tested

Value

A single logical T/F value

is_complete *Get indicators of which dimensions in an ontology are complete*

Description

Get indicators of which dimensions in an ontology are complete

Usage

```
is_complete(x)

is.complete(x)
```

Arguments

ontology	An ontology, as created by ontology

Value

A named logical vector - the corresponding to names(ontology) - indicating whether each dimension is complete

is_complete.ontology *Get indicators of which dimensions in an ontology are complete*

Description

Get indicators of which dimensions in an ontology are complete

Usage

```
## S3 method for class 'ontology'
is_complete(x)
```

Arguments

ontology	An ontology, as created by ontology

Value

A named logical vector - the corresponding to names(ontology) - indicating whether each dimension is complete

JHEEM *Still need to flesh out this interface*

Description

Still need to flesh out this interface

Still need to flesh out this interface

Super class

`jheem2::jheem.entity` -> `jheem`

Methods

Public methods:

- `JHEEM$check()`
- `JHEEM$new()`
- `JHEEM$crunch()`
- `JHEEM$run()`
- `JHEEM$crunch.for.transmutation()`
- `JHEEM$transmute()`
- `JHEEM$test()`
- `JHEEM$extract.quantity.values()`
- `JHEEM$extract.diffeq.settings()`
- `JHEEM$set.element.value()`
- `JHEEM$set.element.functional.form.main.effect.alphas()`
- `JHEEM$set.element.functional.form.interaction.alphas()`
- `JHEEM$set.quantity.foreground()`
- `JHEEM$set.parameters()`
- `JHEEM$set.element.functional.form.from.time()`
- `JHEEM$set.element.functional.form.to.time()`
- `JHEEM$set.element.ramp.times()`
- `JHEEM$set.element.ramp.values()`
- `JHEEM$set.element.taper.times()`
- `JHEEM$set.element.taper.values()`
- `JHEEM$set.element.functional.form.future.slope()`
- `JHEEM$set.element.functional.form.future.slope.after.time()`
- `JHEEM$set.intervention()`
- `JHEEM$set.calibration.code()`
- `JHEEM$has.been.crunched()`
- `JHEEM$clone()`

Method `check()`:

Usage:

```
JHEEM$check()
```

Method `new()`**:**

Usage:

```
JHEEM$new(
  jheem.kernel,
  sub.version,
  transmute.from.kernel,
  transmute.from.sub.version,
  error.prefix = "Cannot create JHEEM Instance: "
)
```

Method `crunch()`**:**

Usage:

```
JHEEM$crunch(
  start.year,
  end.year,
  keep.from.year,
  keep.to.year,
  prior.simulation.set,
  prior.sim.index,
  solver.metadata,
  check.consistency = !self$has.been.crunched()
)
```

Method `run()`**:**

Usage:

```
JHEEM$run(
  start.year,
  end.year,
  check.consistency,
  max.run.time.seconds,
  prior.simulation.set,
  prior.sim.index,
  keep.from.year,
  keep.to.year,
  simulation.chain,
  solver.metadata,
  finalize
)
```

Method `crunch.for.transmutation()`**:**

Usage:

```
JHEEM$crunch.for.transmutation(
  keep.from.year,
  keep.to.year,
```

```
  prior.simulation.set,
  prior.sim.index,
  check.consistency = !self$has.been.crunched()
)
```

Method `transmute()`**:**

Usage:

```
JHEEM$transmute(
  prior.simulation.set,
  prior.sim.index,
  keep.from.year,
  keep.to.year,
  simulation.chain,
  finalize,
  check.consistency
)
```

Method `test()`**:**

Usage:

```
JHEEM$test()
```

Method `extract.quantity.values()`**:**

Usage:

```
JHEEM$extract.quantity.values()
```

Method `extract.diffeq.settings()`**:**

Usage:

```
JHEEM$extract.diffeq.settings()
```

Method `set.element.value()`**:**

Usage:

```
JHEEM$set.element.value(
  element.name,
  value,
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.functional.form.main.effect.alphas()`**:**

Usage:

```
JHEEM$set.element.functional.form.main.effect.alphas(
  element.name,
  alpha.name,
  values,
  dimension,
  applies.to.dimension.values = names(values),
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.functional.form.interaction.alphas()`**:**

Usage:

```
JHEEM$set.element.functional.form.interaction.alphas(
  element.name,
  alpha.name,
  value,
  applies.to.dimension.values,
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.quantity.foreground()`**:**

Usage:

```
JHEEM$set.quantity.foreground(
  foreground,
  check.consistency = !self$has.been.crunched()
)
```

Method `set.parameters()`**:**

Usage:

```
JHEEM$set.parameters(parameters, check.consistency, wrt.version = self$version)
```

Method `set.element.functional.form.from.time()`**:**

Usage:

```
JHEEM$set.element.functional.form.from.time(
  element.name,
  from.time,
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.functional.form.to.time()`**:**

Usage:

```
JHEEM$set.element.functional.form.to.time(
  element.name,
  to.time,
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.ramp.times()`**:**

Usage:

```
JHEEM$set.element.ramp.times(
  element.name,
  times,
  indices = 1:length(times),
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.ramp.values()`**:**

Usage:

```
JHEEM$set.element.ramp.values(
  element.name,
  values,
  indices = 1:length(values),
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.taper.times()`**:**

Usage:

```
JHEEM$set.element.taper.times(
  element.name,
  times,
  indices = 1:length(times),
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.taper.values()`**:**

Usage:

```
JHEEM$set.element.taper.values(
  element.name,
  values,
  indices = 1:length(values),
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.functional.form.future.slope()`**:**

Usage:

```
JHEEM$set.element.functional.form.future.slope(
  element.names,
  slope,
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.element.functional.form.future.slope.after.time()`**:**

Usage:

```
JHEEM$set.element.functional.form.future.slope.after.time(
  element.names,
  after.time,
  check.consistency = !self$has.been.crunched(),
  wrt.version = self$version
)
```

Method `set.intervention()`**:**

Usage:

```
JHEEM$set.intervention(intervention.code)
```

Method `set.calibration.code()`**:**

Usage:

```
JHEEM$set.calibration.code(calibration.code, outcome.location.mapping)
```

Method `has.been.crunched()`**:**

Usage:

```
JHEEM$has.been.crunched()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
JHEEM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`jheem2` *jheem2: A package for setting up and running simulations using the JHEEM Engine #'*

Description

The engine for the JHEEM (Joint HIV Epidemiology and Economic Model) family of compartmental models of HIV and STI transmission. Provides tools to specify versioned model structures, compile and run them as differential-equation simulations, manage surveillance data, map between data ontologies, define interventions, and calibrate models to data via likelihoods.

jheem2 functions

The jheem2 functions ...

`jheem2-calibration-params` *JHEEM Calibration Parameters*

Description

JHEEM Calibration Parameters

jheem2-data-manager-params *JHEEM Data Manager Parameters*

Description

JHEEM Data Manager Parameters

Arguments

metric	The type of measurement. The default value is "estimate", but other options include "cv", "variance", and "sd".
ont	An ontology object as created by ontology()
outcomes	A character vector with the names of outcomes previously registered to this data manager
sources	A character vector with the names of sources previously registered to this data manager

jheem2-likelihood-params *JHEEM Likelihood Parameters*

Description

JHEEM Likelihood Parameters

Arguments

log	Whether to use log likelihood
instructions	A 'jheem.likelihood.instructions' object

jheem2-params *JHEEM Parameters*

Description

JHEEM Parameters

Arguments

version	The version of the model specification (must have been previously registered)
data.manager	A jheem.data.manager object
error.prefix	(Optional) A message to prepend to error messages.
sim	A 'jheem.simulation.set' object
simset	A 'jheem.simulation.set' object
likelihood	A 'jheem.likelihood' object
likelihood.instructions	A 'jheem.likelihood.instructions' object
debug	(dev argument)

---

jheem2-simset-params *JHEEM Simulation Set Parameters*

Description

JHEEM Simulation Set Parameters

Arguments

match.names	A regex to select parameters by name. If NULL, all parameters are used.
chains	Which chains to use, defaulting to all.

---

join.interventions *Join Multiple Interventions into a Single Intervention*

Description

Join Multiple Interventions into a Single Intervention

Usage

```
join.interventions(
  ...,
  code = NULL,
  name = NULL,
  parameters = NULL,
  parameter.distribution = NULL,
  overwrite.existing.intervention = F,
  sequential = F
)
```

Arguments

...	One or more interventions to join. These may be either objects of class jheem.intervention or lists which contain only objects of class jheem.intervention
code	A brief (2-30 character) string that will uniquely identify this intervention
parameter.distribution	
sequential	

---

join.likelihood.instructions

Make a single (joint) set of likelihood instructions out of two or more likelihood instructions objects

---

Description

Make a single (joint) set of likelihood instructions out of two or more likelihood instructions objects

Usage

```
join.likelihood.instructions(..., additional.weights = NULL)
```

Arguments

...	One or more jheem.likelihood.instructions objects or lists which contain only jheem.likelihood.instructions objects #OR it could just be codes
additional.weights	A single numeric value that applies weight to all instructions.

---

join.outcome.location.mappings

Join Outcome Location Mappings

Description

Join Outcome Location Mappings

Usage

```
join.outcome.location.mappings(mappings.to.join)
```

Arguments

mappings.to.join	A list of outcome.location.mappings objects

join.simulation.sets *Join multiple simulation sets into a single set*

Description

Join multiple simulation sets into a single set

Usage

```
join.simulation.sets(...)
```

Arguments

...	One or more jheem.simulation.set objects or lists that contain only jheem.simulation.set objects

Value

A jheem.simulation.set object

list.data.managers *List registered data managers*

Description

List registered data managers

Usage

```
list.data.managers()
```

Value

A data frame with one row per registered name: the name, the data manager's own name, and its file, release, MD5 digest and whether it was modified (NA where unknown)

load.data.manager *Load a cached data manager*

Description

Load a cached data manager

Usage

```
load.data.manager(
  file,
  name = NULL,
  description = NULL,
  set.as.default = F,
  provenance = NULL
)
```

Arguments

file	A file storing a cached data manager object (only)
name	A name for the local data manager. If NULL, the name of the cached data manager.
description	A description for the local data manager. If NULL, the description of the cached data manager.
set.as.default	Should the loaded data manager be set as the default data manager
provenance	Optional: what the caller knows about the file's release, as a named list with any of 'sha256', 'release' and 'repository'. Kept with the file's name and MD5 digest; see get.data.manager.provenance

load.simulation.set *Load a Simulation Set*

Description

Load a Simulation Set

Usage

```
load.simulation.set(file)
```

Arguments

file	The file containing the saved simulation set

make.age.strata.names *Make Names for a Set of Age Strata*

Description

Make Names for a Set of Age Strata

Usage

```
make.age.strata.names(endpoints = NULL, lowers = NULL, uppers = NULL)
```

Arguments

endpoints	A numeric vector of at least two points. endpoints[1] is the lower bound (inclusive) of the first stratum, endpoints[2] is the upper bound (exclusive) for the first stratum and the lower bound for the second stratum, etc.

Value

A character vector with length(endpoints)-1 values

map.value.ontology *Map Value's Ontology*

Description

Map Value's Ontology

Usage

```
map.value.ontology(
  value,
  target.dim.names,
  fun = "sum",
  allow.expand.values = T,
  allow.map.to.subset.of.target = T,
  allow.restratify.ages = T,
  allow.age.extrapolation = F,
  na.rm = T,
  smooth.infinite.age.to = 101,
  error.prefix = ""
)
```

Arguments

allow.map.to.subset.of.target

error.prefix

See Also

Other Functions to identify why ontologies fail to map: apply.ontology.mapping(), get.identity.ontology.mapping()

---

mcmc.traceplot *Create Traceplot of MCMC*

Description

Create Traceplot of MCMC

Usage

```
mcmc.traceplot(simset, match.names, chains = simset$unique.chains, burn = 0)
```

Arguments

match.names	A regex to select parameters by name. If NULL, all parameters are used.
chains	Which chains to use, defaulting to all.
burn	How many iterations to burn and thus ignore on the plot, defaulting to 0.

---

names<-.ontology *Modify an Ontology's Names*

Description

Modify an Ontology's Names

Usage

```
## S3 replacement method for class 'ontology'
names(ont) <- value
```

ontology *Create an ontology object*

Description

Create an ontology object

Usage

```
ontology(..., incomplete.dimensions = NULL)
```

Arguments

`...`	One or more character vectors which must be NAMED
`incomplete.dimensions`	A character vector indicating which dimensions are incomplete. Must be a subset of the names of ...

Details

An ontology is a set of dimnames with specific characteristics: (1) the dimnames must be named, (2) some of the dimensions may be noted as "incomplete". An incomplete dimension is one for which the given values for a dimension may not represent all possible values for that dimension

ONTOLOGY.MAPPING *The Constructor*

Description

The Constructor

The Constructor

The Print Function

A test for equality - to be overwritten in subclass –#

Test if this ontology.mapping can apply to the given dim.names

Test if this ontology.mapping can apply to the given ontology

Get the dimnames that would be generated after applying this mapping to some data with the given dim.names

Get the ontology that would be generated after applying this mapping to some data

Get the dimnames that would be generated after applying this mapping to some data with the given dim.names

Methods

Public methods:

- ONTOLOGY.MAPPING$new()
- ONTOLOGY.MAPPING$print()
- ONTOLOGY.MAPPING$equals()
- ONTOLOGY.MAPPING$can.apply.to.dim.names()
- ONTOLOGY.MAPPING$can.apply.to.ontology()
- ONTOLOGY.MAPPING$apply.to.dim.names()
- ONTOLOGY.MAPPING$apply.to.ontology()
- ONTOLOGY.MAPPING$reverse.apply.to.dim.names()
- ONTOLOGY.MAPPING$apply()
- ONTOLOGY.MAPPING$reverse.apply()
- ONTOLOGY.MAPPING$get.matrix()
- ONTOLOGY.MAPPING$get.mapping.indices()
- ONTOLOGY.MAPPING$get.mapping.vector()
- ONTOLOGY.MAPPING$get.reverse.mapping.vector()
- ONTOLOGY.MAPPING$get.reverse.mapping.indices()
- ONTOLOGY.MAPPING$get.required.from.dim.names()
- ONTOLOGY.MAPPING$get.required.from.dimensions()
- ONTOLOGY.MAPPING$clone()

Method `new()`:

Usage:

```
ONTOLOGY.MAPPING$new(name, component.names = name)
```

Method `print()`:

Usage:

```
ONTOLOGY.MAPPING$print(...)
```

Method `equals()`:

Usage:

```
ONTOLOGY.MAPPING$equals(other)
```

Method `can.apply.to.dim.names()`:

Usage:

```
ONTOLOGY.MAPPING$can.apply.to.dim.names(
  from.dim.names,
  to.dim.names = NULL,
  thorough.validation = T,
  throw.errors = F,
  error.prefix = ""
)
```

Method `can.apply.to.ontology()`:

Usage:

```
ONTOLOGY.MAPPING$can.apply.to.ontology(
  from.ontology,
  to.ontology = NULL,
  thorough.validation = T,
  throw.errors = F,
  error.prefix = ""
)
```

Method `apply.to.dim.names()`**:**

Usage:

```
ONTOLOGY.MAPPING$apply.to.dim.names(
  from.dim.names,
  thorough.validation = T,
  error.prefix = "Cannot apply ontology.mapping to dim names: "
)
```

Method `apply.to.ontology()`**:**

Usage:

```
ONTOLOGY.MAPPING$apply.to.ontology(
  ontology,
  thorough.validation = T,
  error.prefix = "Cannot apply ontology.mapping to ontology: "
)
```

Details: An alias of the apply.to.dim.names function (which appropriately handles if dim.names is additionally an ontology)

Method `reverse.apply.to.dim.names()`**:**

Usage:

```
ONTOLOGY.MAPPING$reverse.apply.to.dim.names(
  to.dim.names,
  error.prefix = "Cannot apply ontology.mapping from dim names: "
)
```

Method `apply()`**:**

Usage:

```
ONTOLOGY.MAPPING$apply(
  from.arr,
  to.dim.names = NULL,
  fun = "sum",
  na.rm = F,
  error.prefix = "Cannot apply ontology.mapping: "
)
```

Method `reverse.apply()`**:**

Usage:

```
ONTOLOGY.MAPPING$reverse.apply(
  to.arr,
  from.dim.names = NULL,
  na.rm = F,
  error.prefix = "Cannot reverse-apply ontology.mapping: "
)
```

Method `get.matrix()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.matrix(
  from.dim.names,
  to.dim.names,
  error.prefix = "Cannot get matrix from ontology.mapping: "
)
```

Method `get.mapping.indices()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.mapping.indices(
  from.dim.names,
  to.dim.names,
  error.prefix = "Cannot get mapping indices from ontology.mapping: "
)
```

Returns: A list, with length the same as the length that would be generated by applying the ontology mapping to data with the given from.dim.names The ith element of the return value is an integer vector containing the indices into the from data which are combined ie - to[i] would be equal to fun(from[get.mapping.indices()[[i]]])

Method `get.mapping.vector()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.mapping.vector(
  from.values = character(),
  from.dimension = self$from.dimensions[1],
  to.dimension = self$to.dimensions[1]
)
```

Method `get.reverse.mapping.vector()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.reverse.mapping.vector(
  to.values = character(),
  from.dimension = self$from.dimensions[1],
  to.dimension = self$to.dimensions[1]
)
```

Method `get.reverse.mapping.indices()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.reverse.mapping.indices(
  from.dim.names,
  to.dim.names,
  error.prefix = "Cannot get mapping indices from ontology.mapping: "
)
```

Method `get.required.from.dim.names()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.required.from.dim.names(to.dim.names)
```

Method `get.required.from.dimensions()`**:**

Usage:

```
ONTOLOGY.MAPPING$get.required.from.dimensions(to.dimensions)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ONTOLOGY.MAPPING$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`outer.join.dim.names` *Join multiple dim.names lists*

Description

Join multiple dim.names lists

Usage

```
outer.join.dim.names(..., error.prefix = "")
```

Arguments

...	can be either (1) dim.names or (2) lists of dim.names

Value

A named list, with an element for each name that appears in any of the dim.names in ... - rv[[d]] is the union of all values in the d dimension of any dim.names in ...

parse.age.strata.names *Convert Age Strata Names into Lower and Upper Bounds for Each Stratum*

Description

Convert Age Strata Names into Lower and Upper Bounds for Each Stratum

Usage

```
parse.age.strata.names(strata.names, allow.partial.parsing = F)
```

Arguments

strata.names	Names of age brackets. The function knows how to parse names generated in the format given by make.age.strata.names, as well as some other common formats
allow.partial.parsing	Whether to parse if only SOME of the age brackets are parseable

Value

A list with two elements, $lowers and $uppers, representing the lower (inclusive) and upper (exclusive) bounds of each age stratum

parse.simset.filename *Parse Simset Filename*

Description

Parses the calibration information that was compressed into the filename of a properly saved calibration simset.

Usage

```
parse.simset.filename(filename, throw.error.if.malformed = T)
```

parse.year.names *Parse Year Names*

Description

Parse Year Names

Usage

```
parse.year.names(years, allow.partial.parsing = F)
```

percentage.cache.complete

Get Percentage of Run Cache Completed

Description

Get Percentage of Run Cache Completed

Usage

```
percentage.cache.complete(version, calibration.code, locations, chains)
```

Value

Matrix of percentages shape [chain x location]

plot.data.validation

Cleaning and verifying the plotting data so it can be properly plotted

Description

Cleaning and verifying the plotting data so it can be properly plotted

Usage

```
## S3 method for class 'data.validation'
plot(
  simset.args,
  deparsed.substituted.args.simset.args,
  outcomes,
  corresponding.data.outcomes,
  plot.which,
  summary.type,
  simset.names = NULL
)
```

Arguments

simset.args	A list of the ... parameter in the previous function, containing simset data. Will be processed and returned as $simset.list
deparsed.substituted.args.simset.args	The ... parameter with each argument converted to a character exactly as it is, without evaluating any expressions. Used to give un-named simsets names later.
outcomes	The outcomes asked for by the plotting function. Will be processed and returned as $outcomes

corresponding.data.outcomes	Checked for proper value, not passed out of the function
plot.which	Checked for proper value, not passed out of the function
summary.type	Checked for proper value, not passed out of the function

Value

A list containing processed $simset.list and $outcomes (a vector)

`plot.simulations` *Plot Simulations With Plotly*

Description

Plot Simulations With Plotly

Usage

```
## S3 method for class 'simulations'
plot(
  ...,
  outcomes,
  corresponding.data.outcomes = NULL,
  split.by = NULL,
  facet.by = NULL,
  dimension.values = list(),
  target.ontology = NULL,
  summary.type = c("individual.simulation", "mean.and.interval",
    "median.and.interval")[1],
  plot.year.lag.ratio = F,
  title = "location",
  n.facet.rows = NULL,
  interval.coverate = 0.95,
  data.manager = get.default.data.manager(),
  style.manager = get.default.style.manager(),
  hide.legend = FALSE
)
```

Arguments

...	One or more of either (1) jheem.simulation.set objects or (2) lists containing only jheem.simulation or jheem.simset objects
outcomes	A character vector of which simulation outcomes to plot
split.by	A character vector of dimensions for which to make different lines
facet.by	A character vector of dimensions for which to make different panels
data.manager	The data.manager from which to draw real-world data for the plots
style.manager	An object of class 'jheem.style.manager' specifying style attributes

Details

See "Simplot" for ggplot2-based plotting.

`prepare.plot` *Prepare Plot Data A utility that performs the data-pulling half of the simplot operation for applications like webtools that need the step to be separate.*

Description

Prepare Plot Data A utility that performs the data-pulling half of the simplot operation for applications like webtools that need the step to be separate.

Usage

```
prepare.plot(
  simset.list = NULL,
  outcomes = NULL,
  locations = NULL,
  corresponding.data.outcomes = NULL,
  split.by = NULL,
  facet.by = NULL,
  dimension.values = list(),
  dimension.values.post.mapping = list(),
  target.ontology = NULL,
  plot.which = c("sim.and.data", "sim.only", "data.only")[1],
  summary.type = c("individual.simulation", "mean.and.interval",
    "median.and.interval")[1],
  omit.data.years = NULL,
  data.na.rm = NULL,
  data.locations = NULL,
  plot.year.lag.ratio = F,
  label.function = NULL,
  title = "location",
  title.suffix = NULL,
  append.url = F,
  data.manager = get.default.data.manager(),
  style.manager = get.default.style.manager(),
  show.data.pull.error = F,
  debug = F
)
```

Arguments

`simset.list`	A list of jheem.simulation.set objects
`outcomes`	A character vector of which simulation outcomes to plot

corresponding.data.outcomes
: Specify directly which data outcomes should be plotted against simulation outcomes. Must be NULL or a character vector with outcomes as names; all of those outcomes must be present in either the 'outcomes' argument or in '...'"

split.by
: At most one dimension

facet.by
: Any number of dimensions but cannot include the split.by dimension

dimension.values.post.mapping
: How to subset data in the form it will take after mappings have been applied to align sim and data ontologies. A list.

plot.which
: Should simulation data and calibration data be plotted ('sim.and.data'), or only simulation data ('sim.only'), or only calibration data ('data.only')

omit.data.years
: Years to omit data, for example if data is available but wasn't calibrated to, while still showing simulation values for those years

data.na.rm
: Sets na.rm = T or F during the data pull step. If NULL, will use T for rates, proportions, and ratios, and F for all other scales.

data.locations
: Specifies locations for which to pull data. May be 1. NULL, in which case data locations will be inferred from the simsets' likelihoods, 2. a character vector of locations, or 3. a list of character vectors with outcomes as list names. For 3., outcomes not included will have their data locations inferred as in 1.

label.function
: A function to reformat labels. If NULL, will use the first provided simset's "get.labels" property, or will do no transformation if plotting data only.

title
: NULL or a single, non-NA character value. If "location", the location of the first provided simset (if any) will be used for the title.

title.suffix
: NULL or a single, non-NA character value. Something to append to the end of the title that would be generated automatically based on the 'title' argument. Note that this pastes with no space, so add the space in if you want it.

data.manager
: The data.manager from which to draw real-world data for the plots

style.manager
: We are going to have to define this down the road. It's going to govern how we do lines and sizes and colors. For now, just hard code those in, and we'll circle back to it

show.data.pull.error
: Not finding data does not block the plot from showing simulation projections, but if you'd like to see in error when data doesn't appear, set this to TRUE.

---

print.ontology *Print an ontology*

Description

Print an ontology

Usage

```
## S3 method for class 'ontology'
print(ont)
```

Arguments

ontology	An ontology, as created by ontology

---

pull.data *Pull data from a data manager*

Description

Pull data from a data manager

Usage

```
pull.data(
  data.manager = get.default.data.manager(),
  outcome,
  metric = "estimate",
  keep.dimensions = NULL,
  dimension.values = NULL,
  sources = NULL,
  from.ontology.names = NULL,
  exclude.ontology.names = NULL,
  target.ontology = NULL,
  allow.mapping.from.target.ontology = T,
  append.attributes = NULL,
  metadata.collapse = " | ",
  append.metrics = NULL,
  allow.other.sources.for.denominator = F,
  na.rm = F,
  ignore.ontologies.without.requested.locations = T,
  check.arguments = T,
  debug = F,
  ...
)
```

Arguments

data.manager	A jheem.data.manager object
outcome	The outcome type for the data. Must be an outcome previously registered to this data manager with register.data.outcome
metric	The type of measurement. The default value is "estimate", but other options include "cv", "variance", and "sd".

keep.dimensions
: The dimensions that should be retained in the returned value

dimension.values
: A named list, indicating which values for specific dimensions to pull data for. Each element must be a named character, numeric, or logical vector.

sources
: The data sources from which to pull data (if available). If NULL, will pull from all data sources that have any relevant data

from.ontology.names
: The names of the ontologies from which to pull data (if available). If NULL, will pull from all ontologies that have any relevant data and can be mapped to the requested ontology. Must refer to ontologies previously registered to this data manager with register.data.ontology

target.ontology
: Optional argument, indicating the ontology according to which results are desired. The data manager will apply an ontology mapping (if it can) to align its data to the desired ontology

allow.mapping.from.target.ontology
: A logical indicator. If TRUE, if target.ontology is specified, but the data manager does not have data that can be mapped to the target ontology, it will search for data such that an ontology.mapping can be applied to data in the target.ontology that make those data align with the data pulled

append.attributes
: A character vector indicating requested data attributes to include with the result. May contain "url", "details", the name of any other registered metadata field, and/or "metadata"

metadata.collapse
: A single character value used to separate multiple metadata values (eg several URLs) within one element of the appended attributes. Defaults to " | ", which cannot appear in a valid URL. Use "__" to get the separator used before this argument existed

na.rm
: Whether to disregard NA values when aggregating out dimensions not in keep.dimensions

ignore.ontologies.without.requested.locations
: If 'dimension.values' includes a location dimension, should ontologies that don't have any of those locations for the requested outcome be ignored? See detailed documentation about how ontologies factor into the output. (NOT YET WRITTEN)

...
: Optional alternative way to specify dimension.values. Each element must be a named character, numeric, or logical vector.

include.sources.without.data.in.output
: When 'sources' is not NULL, include all sources in 'sources' in the output array even if no data was found for some, or even all, sources. If 'sources' is NULL, will include all registered sources for the outcome in the output array.

Value

A numeric array with named dimnames, with one dimension for each value of 'keep.dimensions', plus an additional dimension, "source" at the beginning, with one value for each source from

which data were pulled The return value may have several attributes: (1) If target.ontology is not-NULL and allow.mapping.from.target.ontology==T, 'ontology.mapping' with refer to an ontology.mapping object that can be applied to data in the target.ontology to make them align with the returned array (2) If append.attributes includes "url", an array with the same dimensions as the returned array, where each element is one or more URLs separated by 'metadata.collapse' (3) If append.attributes includes "details", an array with the same dimensions as the returned array, where each element is one or more details about data collection separated by 'metadata.collapse' (4) For any other metadata field named in append.attributes (eg "author"), a character array of the same form as for "url" (5) If append.attributes includes "metadata", 'metadata.ids', a list array with the same dimensions as the returned array where each element is an integer vector of metadata record IDs (NA where the data are NA), and 'metadata', a data frame of the records those IDs refer to (as returned by the data manager's get.metadata.records() method) For all of these, an element is NA wherever the returned data are NA

`put.data` *Put data into a data manager*

Description

Put data into a data manager

Usage

```
put.data(
  data.manager = get.default.data.manager(),
  data,
  outcome,
  metric,
  source,
  ontology.name,
  dimension.values = NULL,
  url = NULL,
  details = NULL,
  metadata = NULL,
  allow.na.to.overwrite = F,
  dimension.values.to.distribute = list(),
  allow.distributing.on.proportion.scale = F,
  round.distributed.dimension.values = T,
  debug = F,
  printouts = F
)
```

Arguments

`data.manager`	A jheem.data.manager object
`data`	A numeric array or scalar value containing the data to store. If it is an array, it must have named dimnames set

`outcome` The outcome type for the data. Must be an outcome previously registered with `register.data.outcome`

`metric` The type of measurement. The default value is "estimate", but other options include "cv", "variance", and "sd".

`source` The (single) character name of the source from which these data derive. Must be a source previously registered with `register.data.source` Note: a source is conceived such that one source cannot contain two sets of data for the same set of dimension values

`ontology.name` The name of the ontology which the data follow. Must be an ontology previously registered with `register.data.ontology`

`dimension.values`
A named list that indicates what subset of a bigger data element these particular data should be stored into. Each element must be a named character, numeric, or logical vector.

`url` A character vector with at least one element giving one or more URLs indicating where the data derive from

`details` A character vector with at least one element giving one or more URLs giving data collection details. In general, data collected and tabulated using the same approach should have the same details, whereas data with different methods/tabulation should have different details

`metadata` A named list of metadata describing every data point in this put, such as list(url=..., details=..., author="CDC", year="2021"). Names must be fields registered with `register.data.metadata.field` ("url" and "details" are always registered). Each element must be a character (or numeric) vector with no NA or empty values. 'url' and 'details' may be given here or in their own arguments, but not both with different values

`allow.na.to.overwrite`
A logical indicator for whether NA values in data should be allowed to overwrite previous values put to the data manager (if data have been put previously)

`dimension.values.to.distribute`
A named list of the dimensions and their values which should be redistributed

`round.distributed.dimension.values`
Should distributed quantities be rounded?

`put.data.long.form` *Put long-form data into a data manager #'@inheritParams distribute.dimension.values*

Description

Put long-form data into a data manager

#'@inheritParams distribute.dimension.values

Usage

```
put.data.long.form(
  data.manager = get.default.data.manager(),
  data,
  outcome,
  metric,
  source,
  ontology.name,
  dimension.values = NULL,
  url = NULL,
  details = NULL,
  metadata = NULL,
  allow.na.to.overwrite = F,
  dimension.values.to.distribute = list(),
  allow.distributing.on.proportion.scale = F,
  round.distributed.dimension.values = T,
  debug = F,
  printouts = F
)
```

Arguments

data.manager	A jheem.data.manager object
data	A data frame or other 2-dimensional data structure with named columns. Must contain a column named 'value' of numeric values. If the 'source' argument is NULL, then data must also have a column named 'source' that gives the source for each row. May contain additional columns with names matching the specified ontology, which have the dimension values for each row. Any other columns are ignored
outcome	The outcome type for the data. Must be an outcome previously registered with register.data.outcome
metric	The type of measurement. The default value is "estimate", but other options include "cv", "variance", and "sd".
source	The (single) character name of the source from which these data derive. Must be a source previously registered with register.data.source Note: a source is conceived such that one source cannot contain two sets of data for the same set of dimension values
ontology.name	The name of the ontology which the data follow. Must be an ontology previously registered with register.data.ontology
dimension.values	A named list that indicates what subset of a bigger data element these particular data should be stored into. Each element must be a named character, numeric, or logical vector.
url	A character vector with at least one element giving one or more URLs indicating where the data derive from

`details` A character vector with at least one element giving one or more URLs giving data collection details. In general, data collected and tabulated using the same approach should have the same details, whereas data with different methods/tabulation should have different details

`metadata` A named list of metadata describing every data point in this put, such as list(url=..., details=..., author="CDC", year="2021"). Names must be fields registered with `register.data.metadata.field` ("url" and "details" are always registered). Each element must be a character (or numeric) vector with no NA or empty values. 'url' and 'details' may be given here or in their own arguments, but not both with different values

`allow.na.to.overwrite`
A logical indicator for whether NA values in data should be allowed to overwrite previous values put to the data manager (if data have been put previously)

`dimension.values.to.distribute`
A named list of the dimensions and their values which should be redistributed

`round.distributed.dimension.values`
Should distributed quantities be rounded?

`rebuild.data.binding` *Rebuild the data binding an artifact was built under*

Description

Loads each recorded data manager through the release resolver (see `set.data.release.resolver`) and checks it against the recorded digests.

Usage

```
rebuild.data.binding(x)
```

Arguments

`x` A record from `get.data.inputs`, or anything get.data.inputs() accepts

Value

A 'jheem.data.binding' object with the recorded names

`register.aging` *Register Information about Aging for a Model Specification*

Description

Register Information about Aging for a Model Specification

Usage

```
register.aging(
  specification,
  aging.rate.value,
  tag = "aging",
  groups = c("uninfected", "infected"),
  applies.to = list()
)
```

Arguments

`specification`	The jheem.specification object to modify
`aging.rate.value`	The values for the aging rate. Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
`tag`	A tag used to follow track this component
`groups`	The group(s) to which the aging apply. Either 'infected' or 'uninfected' or both
`applies.to`	A named list of character or integer vectors, denoting what subset of the group aging applies to

See Also

Other Defining Core Components for a Model Specification: `register.model.mechanism()`, `register.mortality()`, `register.natality()`, `register.remission()`, `register.transition()`, `register.transmission()`

`register.calibrated.parameters.for.version` *Register Parameters to Be Used in Calibrating a JHEEM Version*

Description

Register Parameters to Be Used in Calibrating a JHEEM Version

Usage

```
register.calibrated.parameters.for.version(
  version,
  distribution,
  sampling.blocks,
  apply.function,
  join.with.previous.version,
  calibrate.to.year
)
```

Arguments

`version`	The name of an EHE version for which a specification has been registered with `register.model.specification`
`distribution`	An object of class 'distribution' that specifies the prior distribution for the parameters
`sampling.blocks`	A list (which may optionally be named) of character vectors. Each element (a character vector) denotes a subset of the parameters to be sampled as a single block during the calibration process. These elements must contain ONLY values present in distribution@var.names, and every value in distribution@var.names must be present in at least one of the blocks
`apply.function`	A function that knows how to apply parameter values to a jheem.model.settings object. The function should take two arguments: (1) 'model.settings', an object of class jheem.model.settings, and (2) 'parameters', a named numeric vector
`join.with.previous.version`	Whether the distribution, apply.function, and sampling blocks should be merged with those of the previous version
`calibrate.to.year`	The year to which the calibration should be run before sampling parameters

Details

Calibrated parameters have their values fitted though a Bayesian calibration process

`register.calibration.info` *Register Calibration Info*

Description

Register Calibration Info

Usage

```
register.calibration.info(
  code,
  likelihood.instructions,
  is.preliminary,
  end.year,
  parameter.names,
  parameter.aliases = NULL,
  n.iter,
  thin,
  description,
  special.case.likelihood.instructions = list(),
  fixed.initial.parameter.values = numeric(),
  max.run.time.seconds = Inf,
  solver.metadata = NULL,
  n.chains = if (is.preliminary) 1 else 4,
  n.burn = if (is.preliminary) 0 else floor(n.iter/2),
  data.manager = get.default.data.manager(),
  preceding.calibration.codes = character(),
  draw.from.parent.version = F,
  weight.to.preceding.variance.estimates = 0.5,
  error.prefix = "Error registering calibration info: "
)
```

Arguments

`parameter.aliases`
: A named list of character vectors. Each character vector must be one or more parameter names from the parameter distribution, and the names represent the aliases to be used in sampling

`data.manager`
: The data manager the likelihood is instantiated with, or its name in the data binding. Under a binding, a data manager bound under a name is stored as that name, and the name is resolved in the binding active when the calibration is set up

`error.prefix`

register.data.manager *Register a data manager under a name*

Description

Makes a data manager available as get.data.manager(name). Registering a name again replaces the earlier manager.

Usage

```
register.data.manager(name, data.manager)
```

Arguments

name	A name for the data manager, such as 'census': letters, digits, '.' and '_', starting with a letter
data.manager	A jheem.data.manager object

Value

The data manager, invisibly, so the result can also be assigned

register.data.metadata.field *Register a metadata field to a data manager*

Description

Register a metadata field to a data manager

Usage

```
register.data.metadata.field(
  data.manager = get.default.data.manager(),
  name,
  required = F
)
```

Arguments

data.manager	A jheem.data.manager object
name	The name (a single character value) of the field, such as "author" or "year"
required	A logical indicating whether every subsequent put must supply a value for this field. Data already in the data manager are not affected

Details

Every data manager starts with the fields "url" and "details", both required

---

`register.data.ontology` *Register a new data ontology to a data manager before putting data to it*

Description

Register a new data ontology to a data manager before putting data to it

Usage

```
register.data.ontology(data.manager = get.default.data.manager(), name, ont)
```

Arguments

`data.manager`	A jheem.data.manager object
`name`	The name of the data group
`ont`	An ontology object as created by `ontology()`

`register.data.outcome` *Register an outcome to a data manager before putting data for that outcome*

Description

Register an outcome to a data manager before putting data for that outcome

Usage

```
register.data.outcome(
  data.manager = get.default.data.manager(),
  outcome,
  metadata,
  denominator.outcome = NULL,
  denominator.lags.by.one.year = F,
  overwrite = F,
  allow.missing.denominator.outcome = F
)
```

Arguments

data.manager	A jheem.data.manager object
outcome	The name (a single character value) of the outcome. This is the 'internal' name by which the outcome will be referenced in accessing the data manager
metadata	An object of class 'outcome.metadata', as created by create.outcome.metadata that contains information about how to display the outcome
denominator.outcome	The denominator outcome type that should be used when aggregating data (taking a weighted average) for this outcome type. Must be a previously registered outcome with scale='non.negative.number'. Only applies if scale is 'rate', 'proportion', or 'time'
denominator.lags.by.one.year	Flag indicating whether the denominator data is shifting one year earlier than this outcome's data. For example, data for this outcome in year 2020 will use the 2019 denominator data if this flag is set to TRUE.
overwrite	A logical indicating whether the information on this outcome should overwrite previously-registered information about this outcome. However, this registration must include the same metadata$scale and denominator.outcome (ie, can't change the structure of the outcome, only the display 'trappings')
allow.missing.denominator.outcome	A logical allowing the user to register an outcome of scale 'rate', 'proportion' or 'time' without supplying a denominator outcome

---

register.data.source *Register a data source to a data manager before putting data from that source*

Description

Register a data source to a data manager before putting data from that source

Usage

```
register.data.source(
  data.manager = get.default.data.manager(),
  source,
  parent.source,
  full.name,
  short.name = full.name
)
```

Arguments

data.manager	A jheem.data.manager object
source	The name (a single character value) of the source. This is the 'internal' name by which the source will be referenced in accessing the data manager
parent.source	A parent source registered with this data manager that reflects the origin of the data this source uses
full.name	A descriptive, fully-formatted and capitalized name to use in generating figures and tables (eg in popovers). Avoid abbreviations. Should be unique to the data source (although this is not enforced)
short.name	A name for the data source to use in setting where brevity is important. Ideal to use abbreviations, but can be the same as full.name

Details

Note: a source is conceived such that one source cannot contain two sets of data for the same set of dimension values

---

register.default.parameter.values
Register Default Parameter Values

Description

Register Default Parameter Values

Usage

```
register.default.parameter.values(specification, parameter.values)
```

---

register.fixed.model.strata
Set Whether to Fix Strata Sizes During a Time Period

Description

Set Whether to Fix Strata Sizes During a Time Period

Usage

```
register.fixed.model.strata(
  specification,
  applies.after.time,
  applies.before.time,
  fix.strata = T,
  dimensions.to.fix = NULL
)
```

Arguments

specification	The jheem.specification object
applies.after.time, applies.before.time	Single numeric values giving the time frame over whether this setting applies
fix.strata	A single logical value indicating whether strata should be fixed during this time frame
dimensions.to.fix	A character vector denoting which dimensions should be fixed. These should be dimensions common to both infected and uninfected groups. These are only used if fix.strata==TRUE

---

register.initial.population
Set The Initial Model Population

Description

Set The Initial Model Population

Usage

```
register.initial.population(specification, value, group)
```

Arguments

specification	The jheem.specification object
value	Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
group	Which group ("infected" or "uninfected") this initial population is for

---

register.model.element
Register a Model Element

Description

Register a Model Element

Usage

```
register.model.element(
  specification,
  name,
  scale,
  dimensions = NULL,
  dimension.values = NULL,
  resolve.dimension.values.against.model = T,
  value = NULL,
  get.value.function = NULL,
  functional.form = NULL,
  get.functional.form.function = NULL,
  functional.form.scale = scale,
  functional.form.from.time = NULL,
  functional.form.to.time = Inf,
  get.functional.form.function.name = NULL,
  get.value.function.name = NULL,
  ...,
  ramp.scale = scale,
  ramp.times = numeric(),
  ramp.values = numeric(),
  ramp.value.application = c("multiplier", "absolute")[1],
  ramp.interpolate.links = "identity",
  taper.scale = scale,
  taper.times = numeric(),
  taper.values = numeric(),
  taper.value.application = c("multiplier", "absolute")[1],
  taper.interpolate.links = "identity"
)
```

Arguments

specification	The jheem.specification object
name	The name of the model element. Cannot overlap with the names of model quantities
scale	The scale for this model.element. Can be 'rate', 'ratio', 'proportion', 'time', 'number', 'non.negative.number'
dimensions	Optional parameter to specify which dimensions the model.element is expected to take
dimension.values	Optional parameter to specify which dimension.values the model.element will use in its dimension names
value	A numeric value that serves as the value for this element if it is not otherwise specified. If NULL, then a value must be specified either by a get.value.function (a function which returns a value), or by a functional.form
get.value.function	An alternative to passing value directly, if the value needs to be different by

location. This should be a function that takes arguments location, specification, and ...

`functional.form`
: An object of class 'functional.form' that produces the background values for this element. Pass NULL if no functional.form is to be used, or if it is to be obtained by get.functional.form.function

`get.functional.form.function`
: An alternative to passing background functional.form directly, if the background functional.form needs to be different by location. This should be a function that takes arguments location, specification, and ...

`functional.form.scale`
: The scale of values produced by the background functional.form

`functional.form.from.time`, `functional.form.to.time`
: The time frame over which the functional.form applies

`get.value.function.name`, `get.functional.form.function.name`
: Optional - the actual names of the functions passed to 'get.value.function' or 'get.functional.form.function' - used to make error messages more informative

`...`
: Arguments to be passed to either get.functional.form.function or get.value.function

`ramp.scale`, `taper.scale`
: If the ramp or taper operates on a different scale than the element evaluates to, specified here

`ramp.times`, `taper.times`
: The (default) times at which to ramp up to taper down

`ramp.values`, `taper.values`
: The (default) multipliers for ramp or taper times

`ramp.value.application`, `taper.value.application`
: How to interpret ramp and taper values. 'multiplier' multiplies the ramp by the first functional.form-generated value (and taper by the last functional.form-generated value). 'absolute' just plugs in the value directly

`ramp.interpolate.links`, `taper.interpolate.links`
: The name(s) of link function (either 'identity', 'log', or 'exp') denoting the scale at which to interpolate values for the ramp or taper. Can be either a single value (applied to all elements in the ramp), or a vector of length(ramp.times) or length(taper.times)

`resolve.dimension.values.against.model.ontologies`
: A logical indicating whether, if dimension.values are supplied, they should be resolved against model ontologies and have compartment.value.aliases for the model specification applied to them

---

`register.model.element.values`

A convenience function to streamline registering multiple model.elements for which we only need to specify a value

---

Description

A convenience function to streamline registering multiple model.elements for which we only need to specify a value

Usage

```
register.model.element.values(specification, ..., scale)
```

Arguments

specification	The jheem.specification object to modify
...	A set of named values. The names correspond to the names of the model elements, and the values should be either (1) numeric objects or (2) functions that take arguments 'location' and 'settings' and returns a numeric value
scale	The scale at which the elements apply

`register.model.foreground`

Register a Foreground for a Model Specification

Description

Register a Foreground for a Model Specification

Usage

```
register.model.foreground(specification, foreground, name)
```

Arguments

specification	The jheem.specification object to modify
foreground	An object of class 'jheem.model.foreground', as created by `create.model.foreground`
name	A single character value giving the name by which the foreground can be referred to

`register.model.mechanism` *Register Information about Disease Transmission for a Model Specification*

Description

Register Information about Disease Transmission for a Model Specification

Usage

```
register.model.mechanism(
  specification,
  type,
  value,
  tags,
  groups,
  from.groups,
  to.groups,
  dimension,
  from.compartments,
  to.compartments,
  na.replacement = as.numeric(NA),
  ...
)
```

Arguments

`type`	The type of the model mechanism. Must be one of 'mortality.rate', 'fertility.rate', 'birth.proportions', 'aging.rate', 'transition.rate', 'susceptibility', 'transmissibility', 'contact', 'new.infection.proportions', 'remission.rate', 'remission.proportions'
`dimension`	The name of the dimension in which this transition operates
`from.compartments, to.compartments`	The compartments from and to which this transition applies. May be (1) one or more names of compartments, (2) one or more character compartment.value.aliases, (3) one or more integers denoting compartments, (4) a logical vector, which, when applied to the dimension names, denotes the compartments
`group`	The group(s) to which this transition applies. Either 'uninfected', 'infected', or both

See Also

Other Defining Core Components for a Model Specification: `register.aging()`, `register.mortality()`, `register.natality()`, `register.remission()`, `register.transition()`, `register.transmission()`

`register.model.quantity` *Register a Model Quantity*

Description

Register a Model Quantity

Usage

```
register.model.quantity(
  specification,
  name,
  value,
  scale = NULL,
  dimensions = names(dimension.values),
  dimension.values = NULL,
  resolve.dimension.values.against.model = T,
  ...,
  na.replacement = as.numeric(NA)
)
```

Arguments

`specification`	The jheem.specification object to modify
`name`	The name of the model quantity. Cannot overlap with names of model elements
`value`	Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
`scale`	An optional parameter, indicating the scale for this model.quantity. If specified, running the model will throw an error if evaluating the quantity results in an invalid value for the scale (eg a negative rate or a proportion > 1). Can be 'rate', 'ratio', 'proportion', 'time', 'number', 'non.negative.number'
`dimensions`	An optional parameter, indicating the dimensions the quantity is expected to have (if dim.names is not specified, the specific values of the dimensions will be inferred automatically)
`dimension.values`	An optional parameter, indicating (some of) the elements that the dimnames of the quantity are supposed to contain. This must be a named list of character vectors (whose names are a subset of dimensions if dimensions are specified).
`...`	Additional parameters to be passed to value if a function is passed as the value

na.replacement	A single numeric value that should be used to replace NA values generated in calculating the quantity's value (note, if na.replacement==NA, no handling of NAs is performed and NAs will cause an error in model execution)

`register.model.quantity.subset` *Register a Value to a Subset of a Model Quantity or Transition*

Description

Register a Value to a Subset of a Model Quantity or Transition

Usage

```
register.model.quantity.subset(
  specification,
  name,
  value,
  applies.to,
  na.replacement = as.numeric(NA),
  apply.function = c("overwrite", "add", "subtract", "divide", "multiply")[1],
  ...
)
```

Arguments

specification	The jheem.specification object to modify
name	The name of the model quantity or transition to which to register a subset
value	Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
applies.to	A named list representing the subset of value in question. The names of applies.to should correspond to dimensions of value, and the values must be either character, integer, or logical vectors accessing values in each dimension
na.replacement	A single numeric value that should be used to replace NA values generated in calculating the quantity's value (note, if na.replacement==NA, no handling of NAs is performed and NAs will cause an error in model execution)
apply.function	A character value indicating how the value for this subset should be incorporated into the main quantity. Options are 'overwrite' (the value for the subset overwrites the value in the quantity), 'add' (subset value is added to the quantity), 'subtract' (subset value subtracted from the quantity), 'multiply' (subset value multiplied by the quantity), or 'divide' (quantity is divided by the subset value)
...	Additional parameters to be passed to value if a function is passed as the value

register.model.specification *Register a JHEEM Model Specification*

Description

Register a JHEEM Model Specification

Usage

```
register.model.specification(specification)
```

Arguments

specification	A jheem.specification object, as created by create.jheem.specification

Details

Once a specification object is registered, it can no longer be modified

See Also

Other JHEEM Version Management Functions: get.prior.versions(), get.specification.for.version(), version.has.been.registered()

register.mortality *Register Information about Mortality for a Model Specification*

Description

Register Information about Mortality for a Model Specification

Usage

```
register.mortality(
  specification,
  mortality.rate.value,
  groups = c("uninfected", "infected"),
  tag = "mortality",
  applies.to = list()
)
```

Arguments

specification	The jheem.specification object to modify
mortality.rate.value	The mortality.rate. Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
groups	The group(s) to which the mortality apply. Either 'infected' or 'uninfected' or both
tag	A tag used to follow track this component
applies.to	A named list of character or integer vectors, denoting what subset of the groups the component apply to

See Also

Other Defining Core Components for a Model Specification: register.aging(), register.model.mechanism(), register.natality(), register.remission(), register.transition(), register.transmission()

---

register.natality *Register Information about Natality (Births) for a Model Specification*

Description

Register Information about Natality (Births) for a Model Specification

Usage

```
register.natality(
  specification,
  parent.groups,
  child.groups,
  fertility.rate.value,
  birth.proportions.value,
  parent.child.concordant.dimensions = character(),
  all.births.from.compartments = list(),
  all.births.into.compartments = character(),
  tag = "natality",
  applies.to = list()
)
```

Arguments

specification	The jheem.specification object to modify
parent.groups	The group representing parents (infected or uninfected)
child.groups	The group into which births are distributed (infected or uninfected)
fertility.rate.value, birth.proportions.value	The values for the fertility rate and the proportions by which births are distributed into the to.group. Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
parent.child.concordant.dimensions	A character vector of the dimensions for which all new births share the same value as their parents
all.births.from.compartments	A named list of character or integer values. The names represent dimensions, and the elements represent the compartments from those dimensions that can give birth
all.births.into.compartments	A named list of single character or integer values. The names represent dimensions and the elements represent the single compartment in that dimension into which all births go
tag	A tag used to follow track this component
applies.to	A named list of character or integer vectors, denoting what subset of the from.group fertility applies to

See Also

Other Defining Core Components for a Model Specification: register.aging(), register.model.mechanism(), register.mortality(), register.remission(), register.transition(), register.transmission()

---

register.ontology.mapping

Create and register a mapping that allows transformations of arrays

---

Description

Create and register a mapping that allows transformations of arrays

Usage

```
register.ontology.mapping(
  name,
  mappings,
  from.dimensions,
  to.dimensions = from.dimensions
)
```

Arguments

name	A descriptive single character value
mappings	A character matrix, with a number of columns == length(from.dimensions) + length(to.dimensions). The first length(from.dimensions) columns represent values for the from.dimensions (in the same order as from.dimensions), and the last length(to.dimensions) colums represent values for to.dimensions (in the order of to.dimensions). Each row represents a mapping from a set of values for from.dimension to a set of values for to.dimensions. NB: any dimnames on mappings will be ignored
from.dimensions	A character vector of names of the dimensions we are mapping from
to.dimensions	A character vector of names of the dimensions we are mapping to

Details

The mappings must contain every possible combination of 'from' values (ie, every combo of from values must map to something). A row of 'mappings' may have NA values in ALL the 'from' columns (indicating there is no set of 'from' values that maps to the corresponding 'to' values), or may have NA values in ALL the 'to' columns (indicating the that the given set of 'from' values will be discarded in the transformation)

---

register.parent.data.source

Register a parent data source to a data manager

Description

Register a parent data source to a data manager

Usage

```
register.parent.data.source(
  data.manager = get.default.data.manager(),
  parent.source,
  full.name,
  short.name = full.name
)
```

Arguments

data.manager	A jheem.data.manager object
parent.source	The name (a single character value) of the parent source. This is the 'internal' name by which the parent source will be referenced in accessing the data manager
full.name	A descriptive, fully-formatted and capitalized name to use in generating figures and tables (eg in popovers). Avoid abbreviations. Should be unique to the parent data source (although this is not enforced)
short.name	A name for the parent data source to use in setting where brevity is important. Ideal to use abbreviations, but can be the same as full.name

---

register.remission *Register Information about Remission from Infection for a Model Specification*

Description

Register Information about Remission from Infection for a Model Specification

Usage

```
register.remission(
  specification,
  remission.rate.value,
  remission.proportions.value,
  tag = "remission",
  applies.to = list(),
  all.remissions.into.compartments = character()
)
```

Arguments

specification	The jheem.specification object to modify
remission.rate.value, remission.proportions.value	The values for the remission rate and the proportions by which remitted cases are distributed into the uninfected group. Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
tag	A tag used to follow track this component
applies.to	A named list of character or integer vectors, denoting what subset of the infected group remissions apply to

See Also

Other Defining Core Components for a Model Specification: `register.aging()`, `register.model.mechanism()`, `register.mortality()`, `register.natality()`, `register.transition()`, `register.transmission()`

`register.sampled.parameters.for.version`
Register Parameters to Be Sampled Prior to Making Projections from a JHEEM Version

---

Description

Register Parameters to Be Sampled Prior to Making Projections from a JHEEM Version

Usage

```
register.sampled.parameters.for.version(
  version,
  distribution,
  apply.function,
  join.with.previous.version
)
```

Arguments

`version`	The name of an EHE version for which a specification has been registered with `register.model.specification`
`distribution`	An object of class 'distribution' that specifies the prior distribution for the parameters
`apply.function`	A function that knows how to apply parameter values to a jheem.model.settings object. The function should take two arguments: (1) 'model.settings', an object of class jheem.model.settings, and (2) 'parameters', a named numeric vector
`join.with.previous.version`	Whether the distribution and apply.function should be merged with those of the previous version

Details

Sampled parameters have their values randomly chosen from a distribution prior to running projections PAST the calibration period

register.set.parameters.for.version *Register Parameters to Be Set for Running a Simulation*

Description

Register Parameters to Be Set for Running a Simulation

Usage

```
register.set.parameters.for.version(
  version,
  parameter.names,
  apply.function,
  join.with.previous.version
)
```

Arguments

version	The name of an EHE version for which a specification has been registered with register.model.specification
parameter.names	A character vector giving the names of parameters which will be set
apply.function	A function that knows how to apply parameter values to a jheem.model.settings object. The function should take two arguments: (1) 'model.settings', an object of class jheem.model.settings, and (2) 'parameters', a named numeric vector
join.with.previous.version	Whether the parameter.names and apply.function should be merged with those of the previous version

Details

Set parameters have their values set explicitly (neither sampled nor calibrated)

register.transition *Register Information about Disease Transmission for a Model Specification*

Description

Register Information about Disease Transmission for a Model Specification

Usage

```
register.transition(
  specification,
  dimension,
  from.compartments,
  to.compartments,
  value,
  groups,
  applies.to = list(),
  tag = "transition",
  na.replacement = as.numeric(NA),
  ...
)
```

Arguments

dimension	The name of the dimension in which this transition operates
from.compartments, to.compartments	The compartments from and to which this transition applies. May be (1) one or more names of compartments, (2) one or more character compartment.value.aliases, (3) one or more integers denoting compartments, (4) a logical vector, which, when applied to the dimension names, denotes the compartments
groups	The group(s) to which this transition applies. Either 'uninfected', 'infected', or both

See Also

Other Defining Core Components for a Model Specification: register.aging(), register.model.mechanism(), register.mortality(), register.natality(), register.remission(), register.transmission()

---

register.transmission *Register Information about Disease Transmission for a Model Specification*

Description

Register Information about Disease Transmission for a Model Specification

Usage

```
register.transmission(
  specification,
  contact.value,
  susceptibility.value,
  transmissibility.value,
  new.infection.proportions.value,
```

```
  tag = "transmission",
  all.new.infections.into.compartments = list(),
  from.applies.to = list(),
  to.applies.to = list(),
  transmission.applies.to = list(),
  new.infections.applies.to = list()
)
```

Arguments

specification	The jheem.specification object to modify
contact.value, susceptibility.value, transmissibility.value, new.infection.proportions.value	The values for the contact matrix, susceptibility, transmissibility, and proportions by which new infections are distributed into the infected group. Either: (1) a numeric object, (2) a character string giving the name of a model quantity or a model element, (3) an 'call' or 'expression' object (generated by expression() or expr()) containing an expression comprising other model quantities or model elements, and the operators +, -, *, /, ^, sqrt, log, and exp, or (4) a function whose arguments are some subset of 'specification.metadata', 'location', other model quantities and model elements, and ...
tag	A tag used to follow track this component
from.applies.to	A named list of character or integer vectors, denoting what subset of the population PARTNERS may come from (regardless of whether or not they actually transmit). Must be a subset of compartments.for.infected.and.uninfected
to.applies.to	A named list of character or integer vectors, denoting what subset of the uninfected group are susceptible to infection. Must be a subset of compartments.for.infected.and.uninfected + compartments.for.uninfected.only
transmission.applies.to	A named list of character or integer vectors, denoting what subset of the infected group may transmit infection. Must be a subset of compartments.for.infected.and.uninfected + compartments.for.infected.only
new.infections.applies.to	A named list of character or integer vectors, denoting what subset of the infected group new infections may enter into. Must be a subset of compartments.for.infected.only

See Also

Other Defining Core Components for a Model Specification: register.aging(), register.model.mechanism(), register.mortality(), register.natality(), register.remission(), register.transition()

`register.transmute.calibration.info` *Register Transmute Calibration Info*

Description

Register Transmute Calibration Info

Usage

```
register.transmute.calibration.info(
  transmute.code,
  from.version,
  to.version,
  n.iter.first.sim,
  n.iter.subsequent.sims,
  likelihood.instructions,
  prior.distribution,
  sampling.blocks,
  fixed.initial.parameter.values = numeric(),
  from.sub.version = NULL,
  to.sub.version = NULL,
  max.lookback.attempts = 20
)
```

`remove.data` *Remove Data From Data Manager*

Description

Replaces data with NA if data exists

Usage

```
remove.data(
  data.manager,
  outcome,
  source,
  ontology.name,
  dimension.values,
  metric = "estimate",
  details.for.removal = "removed",
  url.for.removal = "url"
)
```

Arguments

data.manager	A jheem.data.manager object
outcome	The outcome type for the data. Must be an outcome previously registered with register.data.outcome
source	The (single) character name of the source from which these data derive. Must be a source previously registered with register.data.source Note: a source is conceived such that one source cannot contain two sets of data for the same set of dimension values
ontology.name	The name of the ontology which the data follow. Must be an ontology previously registered with register.data.ontology
dimension.values	A named list that indicates what subset of a bigger data element these particular data should be stored into. Each element must be a named character, numeric, or logical vector.
metric	The type of measurement. The default value is "estimate", but other options include "cv", "variance", and "sd".
details.for.removal, url.for.removal	Single character values to use for these points' 'details' and 'url'

---

rerun.simulations *Rerun a Simulation Set*

Description

Rerun a Simulation Set

Usage

```
rerun.simulations(
  simset,
  max.run.time.seconds = Inf,
  sub.version = simset$sub.version,
  from.year = simset$from.year,
  to.year = simset$to.year,
  verbose = T
)
```

Arguments

simset	The JHEEM simulation set to re-run
max.run.time.seconds	The maximum amount of time to run simulations before terminating
sub.version	The sub-version to run
from.year, to.year	The years to keep in the simulation object
verbose	Whether to print updates as simulations are run

Details

Helpful if the specification has changed

`resolve.intervention.effect` *#@export*

Description

#@export

Usage

```
resolve.intervention.effect(intervention.effect, bindings, error.prefix = "")
```

`resolve.ontology.dimension.values` *Resolve dimension.values into a set of dimnames for an ontology*

Description

Resolve dimension.values into a set of dimnames for an ontology

Usage

```
resolve.ontology.dimension.values(
  ont,
  dimension.values,
  error.prefix,
  throw.error.if.unresolvable = T
)
```

Arguments

`ont`	An ontology, as created by `ontology`
`dimension.values`	A list of dimension.values (ie, a named list of either character, integer, or logical vectors that indexes into the dimnames given in ont)
`error.prefix`	Text to prepend to any error messages that are thrown
`throw.error.if.unresolvable`	A logical indicating what to do if unable to resolve against the ontology. When FALSE, will return NULL if unresolvable

Value

If the dimension.values can be resolved against ontology ont, returns a list of dimnames (ie, a named list of character vectors). If cannot resolve will either throw an error (if throw.error.if.unresolvable) or return NULL (NB, if passed dimension.values==NULL, will also return NULL)

`restratify.age.counts` *Re-distribute counts according to one set of age brackets into a different set of age brackets*

Description

Re-distribute counts according to one set of age brackets into a different set of age brackets

Usage

```
restratify.age.counts(
  counts,
  desired.age.brackets,
  given.age.brackets = NULL,
  smooth.infinite.age.to = Inf,
  allow.extrapolation = F,
  na.rm = F,
  method = c("pclm", "monoH.FC", "hyman")[2],
  error.prefix = ""
)
```

Arguments

`counts`	Either (a) a dimensionless numeric vector, or (b) a numeric array, with one dimension named "age". The "given" age brackets according to which these counts are distributed must be continuous - with no gaps between them
`desired.age.brackets, given.age.brackets`	The age brackets according to which the ages in counts are given or desired to be restratified to. Either (1) a character vector giving names of brackets, with one name for each age bracket value or (b) a numeric vector of age endpoints, with one more element than the number of age bracket values. given.age.brackets should be present only if names(counts) is absent (if counts is a vector) or if dimnames(counts)$age is null (if counts is an array)
`smooth.infinite.age.to`	In building a spline over counts, what value should an infinite bound in an age range be replaced with
`allow.extrapolation`	A logical value indicating whether extrapolation should be allowed in restratifying. Extrapolating is required when one of the desired age brackets includes ages not present in any of the given age brackets
`na.rm`	Whether NAs should be ignored in counts

method	The method to be passed to the smoother. Options are 'pclm' (default, uses Penalized Composite Link Model), 'monoH.FC' (Fritsch-Carlson monotonic splines), or 'hyman' (Hyman's monotonic splines)
error.prefix	A character value to be prepended to any error messages

---

retrieve.simulation.set *Retrieve a Saved Simulation Set*

Description

Loads a simulation set that has been properly saved in JHEEM's calibration file structure.

Usage

```
retrieve.simulation.set(
  version,
  location,
  calibration.code,
  n.sim,
  intervention.code = NULL,
  sub.version = NULL,
  root.dir = get.jheem.root.directory("Cannot get.simset.filename(): "),
  error.prefix = "",
  throw.error.if.missing = T
)
```

---

run.calibration *Run A Calibration*

Description

Run A Calibration

Usage

```
run.calibration(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory("Cannot set up calibration: "),
  chains,
  update.frequency = 500,
  update.detail = "low"
)
```

Arguments

version	The version of the model specification (must have been previously registered)
location	A single character value representing the location for the calibration
update.detail	

Value

An MCMC object

---

run.simulations.from.parameters

Run a Simulations from a Set of Parameters

Description

Run a Simulations from a Set of Parameters

Usage

```
run.simulations.from.parameters(
  version,
  location,
  parameters,
  end.year,
  sub.version = NULL,
  max.run.time.seconds = Inf,
  calibration.code = NULL,
  solver.metadata = create.solver.metadata(),
  keep.from.year = start.year,
  keep.to.year = end.year,
  verbose = F
)
```

Arguments

version, location, sub.version	The JHEEM version, sub.version, and location for which to run simulations
parameters	The parameters to run simulations for. Can be either (a) a matrix with parameters on the rows, and one column for each set of parameters/simulation to run, with named rows or (b) a named numeric vector of parameters for a single simulation
end.year	The year to run simulations to
max.run.time.seconds	The maximum amount of time to run simulations before terminating
calibration.code	If parameters were derived from a previous calibration, the calibration code that should be attached to the resulting simulation set

`solver.metadata`
: A solver.metadata object, as created by `create.solver.metadata`, specifying what ODE solver to use in running simulations

---

`run.transmute.calibration` *Run Transmute Calibration*

Description

Run Transmute Calibration

Usage

```
run.transmute.calibration(
  transmute.code,
  location,
  n.sim,
  chunks = NULL,
  verbose = T,
  update.frequency = 500,
  update.detail = "low",
  ignore.errors = T,
  rerun = T,
  root.dir = get.jheem.root.directory()
)
```

`save.simulation.set` *Save a JHEEM Simulation Set*

Description

Save a JHEEM Simulation Set

Usage

```
save.simulation.set(
  simset,
  root.dir = get.jheem.root.directory("Cannot save simulation set: ")
)
```

Arguments

`simset`	A jheem.simulation.set object
`root.dir`	The directory in which the folder tree for saving the simset will be rooted

set.data.binding *Set the session's data binding*

Description

Registers each data manager in the binding under its name, replacing the registry's contents (see `register.data.manager`), and makes the registry the active binding for the session. Data managers registered later join it.

Usage

```
set.data.binding(binding)
```

Arguments

`binding`	A 'jheem.data.binding' object, or NULL to stop using a session binding (the registry is kept)

Value

The binding, invisibly

set.data.release.resolver *Set the function that loads a data manager release*

Description

jheem2 does not fetch releases. `rebuild.data.binding` calls this function to load the data manager each recorded name was bound to.

Usage

```
set.data.release.resolver(resolver)
```

Arguments

`resolver`	A function taking one argument, a list with data.manager.name, file, release, repository, sha256 and md5 (NA where not recorded), and returning the data manager loaded with its provenance (see `load.data.manager`); or NULL to remove it

Value

NULL, invisibly

set.default.data.manager *Set the default data manager*

Description

Set the default data manager

Usage

```
set.default.data.manager(data.manager)
```

set.default.style.manager *Set the Default Style Manager*

Description

Set the Default Style Manager

Usage

```
set.default.style.manager(style.manager, type = c("ggplot", "plotly")[1])
```

set.element.functional.form.alphas.from.parameters *Set multiple functional form main effect alphas from parameters*

Description

Set multiple functional form main effect alphas from parameters

Usage

```
set.element.functional.form.alphas.from.parameters(
  model.settings,
  element.name,
  alpha.name,
  parameters,
  parameter.name.prefix,
  parameter.name.suffix,
  dimensions.with.values.referred.to.by.name = character(),
  dimensions.with.values.referred.to.by.index = character(),
  throw.error.if.no.parameters = T,
  wrt.version = model.settings$version
)
```

Arguments

`model.settings`	An object of class 'jheem.model.settings'
`element.name`	The name of the model element whose functional form we wish to modify
`alpha.name`	The name of the functional form parameter that we want to set alphas for
`parameters`	A named numeric vector with values for interaction terms
`parameter.name.prefix, parameter.name.suffix`	Character values to be prepended and addended to names of specific dimension values to generate specific parameter names
`dimensions.with.values.referred.to.by.name`	Dimensions for which we should look for main effects in parameters, and make parameter names by combining parameter.name.prefix and parameter.name.suffix with the actual dimension values for the dimension
`dimensions.with.values.referred.to.by.index`	Dimensions for which we should look for main effects in parameters, and make parameter names by combining parameter.name.prefix and parameter.name.suffix with indices of the values in each dimension
`throw.error.if.no.parameters`	A logical indicator - if no parameter names match the pattern parameter.name.prefix<x>parameter.name.su whether an error should be thrown
`wrt.version`	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)

Details

This function will search for parameters with names in the format of either parameter.name.prefix<dimension values for dimensions.with.values.referred.to.by.name>parameter.name.suffix OR parameter.name.prefix<dimension in dimension.values.referred.to.by.name><1:length of dimension>parameter.name.suffix

Value

The names of parameters that were used

`set.element.functional.form.from.time`

Set the Times From and To Which the Functional Form Determines a Model Element's Value

---

Description

Set the Times From and To Which the Functional Form Determines a Model Element's Value

Usage

```
set.element.functional.form.from.time(
  model.settings,
  element.name,
  from.time,
  wrt.version = model.settings$version
)

set.element.functional.form.to.time(
  model.settings,
  element.name,
  to.time,
  wrt.version = model.settings$version
)
```

Arguments

`model.settings`	An object of class 'jheem.model.settings'
`element.name`	The name of the model element to set functional.form times for
`from.time, to.time`	The time from or to which the functional form is active
`wrt.version`	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)

Functions

- `set.element.functional.form.to.time()`:

See Also

Other Functions to create and modify a model settings: `set.element.functional.form.future.slope()`, `set.element.functional.form.interaction.alphas()`, `set.element.ramp.times()`

---

set.element.functional.form.future.slope

Set a "Future Slope" and the time after which it applies, for a model element

---

Description

Set a "Future Slope" and the time after which it applies, for a model element

Usage

```
set.element.functional.form.future.slope(
  model.settings,
  element.names,
  slope,
  wrt.version = model.settings$version
)

set.element.functional.form.future.slope.after.time(
  model.settings,
  element.names,
  after.time,
  wrt.version = model.settings$version
)
```

Arguments

model.settings	An object of class 'jheem.model.settings'
element.names	One or more names of the model element to set the future.slope for
slope	A single numeric value representing the slope
wrt.version	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)
after.time	The year after which the additional slope takes effect

Details

A future slope is applied to model elements with a functional.form set AFTER a given point in time

Functions

- set.element.functional.form.future.slope.after.time():

See Also

Other Functions to create and modify a model settings: set.element.functional.form.from.time(), set.element.functional.form.interaction.alphas(), set.element.ramp.times()

---

set.element.functional.form.interaction.alphas

Set Interaction Alpha Values for a Functional Form for a Model Element

---

Description

Set Interaction Alpha Values for a Functional Form for a Model Element

Usage

```
set.element.functional.form.interaction.alphas(
  model.settings,
  element.name,
  alpha.name,
  value,
  applies.to.dimension.values,
  wrt.version = model.settings$version
)
```

Arguments

model.settings	An object of class 'jheem.model.settings'
element.name	The name of the model element whose functional form we wish to modify
alpha.name	The name of the functional form parameter that we want to set alphas for
value	A single numeric value, which will apply to every combination of applies.to.dimension.values which are from different dimensions
applies.to.dimension.values	Either a character vector where each element is either a character vector. Must be named with the dimension to which each dimension value applies
wrt.version	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)

Details

Alphas for an 'interaction' will apply to every combination of applies.to.dimension.values from different dimensions (as opposed to 'main effect' apply to all values in a dimension

See Also

Other Functions to create and modify a model settings: set.element.functional.form.from.time(), set.element.functional.form.future.slope(), set.element.ramp.times()

---

set.element.functional.form.main.effect.alphas

Set Main Effect Alpha Values for a Functional Form for a Model Element

---

Description

Set Main Effect Alpha Values for a Functional Form for a Model Element

Usage

```
set.element.functional.form.main.effect.alphas(
  model.settings,
  element.name,
  alpha.name,
  values,
  dimension,
  applies.to.dimension.values = names(values),
  wrt.version = model.settings$version
)
```

Arguments

model.settings	An object of class 'jheem.model.settings'
element.name	The name of the model element whose functional form we wish to modify
alpha.name	The name of the functional form parameter that we want to set alphas for
values	A numeric vector of values (either a single value applied to all dimension values OR a vector of the same length as applies.to.dimension.values)
dimension	The dimension to which values in applies.to.dimension.values apply
applies.to.dimension.values	A character vector of values within the given dimension to which the alphas should apply
wrt.version	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)

Details

Alphas for a 'main effect' apply to all values in a dimension (as opposed to 'interaction effects', which apply to combinations of values)

---

set.element.ramp.times

Set a ramp or taper (times and values) for a model element

Description

Set a ramp or taper (times and values) for a model element

Usage

```
set.element.ramp.times(
  model.settings,
  element.name,
  times,
```

```
  indices = 1:length(times),
  wrt.version = model.settings$version
)

set.element.ramp.values(
  model.settings,
  element.name,
  values,
  indices = 1:length(values),
  wrt.version = model.settings$version
)

set.element.taper.times(
  model.settings,
  element.name,
  times,
  indices = 1:length(times),
  wrt.version = model.settings$version
)

set.element.taper.values(
  model.settings,
  element.name,
  values,
  indices = 1:length(values),
  wrt.version = model.settings$version
)
```

Arguments

model.settings	An object of class 'jheem.model.settings'
element.name	The name of the model element to set the ramp/taper for
times, values	The times or values to set for the ramp/taper
indices	The indices of the ramp/taper to set times or values for. Can be either numeric indices, character indices (if the ramp/taper was named when setting up the specification), or logical indices
wrt.version	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)

Details

A ramp is a set of multipliers which are multiplied by the first functional.form value (ie, the value produced at functional.form.from.time) for an element. A taper is a set of multipliers which are multiplied by the last functional.form value (ie, the value produced at functional.form.to.time) for an element

Functions

- set.element.ramp.values():
- set.element.taper.times():
- set.element.taper.values():

See Also

Other Functions to create and modify a model settings: set.element.functional.form.from.time(), set.element.functional.form.future.slope(), set.element.functional.form.interaction.alphas()

---

set.element.value *Set the Value of a Model Element*

Description

Set the Value of a Model Element

Usage

```
set.element.value(
  model.settings,
  element.name,
  value,
  wrt.version = model.settings$version
)
```

Arguments

model.settings	An object of class 'jheem.model.settings'
element.name	The name of the model element to update a value for
value	The new value for the model element
wrt.version	The version relative to which the element name is resolved (if an element.name is overwritten in a descendant version, using wrt.version can be used to select which specification's model element is to be modified)

---

set.jheem.root.directory *Set JHEEM Root Directory*

Description

Set the directory that calibration directories and files will be written into.

Usage

```
set.jheem.root.directory(dir)
```

set.up.calibration *Set Up Calibration*

Description

Set Up Calibration

Usage

```
set.up.calibration(
  version,
  location,
  calibration.code,
  root.dir = get.jheem.root.directory("Cannot set up calibration: "),
  sub.version = NULL,
  cache.frequency = 500,
  allow.overwrite.cache = F,
  verbose = T
)
```

Arguments

version	The version of the model specification (must have been previously registered)
location	A single character value representing the location for the calibration
verbose	

Details

The calibration's control saves the functions a chain calls. By default they keep everything this function built, including the calibration's data manager. With `options(jheem2.lean.calibration.controls = TRUE)` they keep only what a chain uses (the engine and the instantiated likelihood, which holds the data it pulled), and the data manager is kept only if it was not loaded from a release. Chains are the same either way.

set.up.transmute.calibration *Set Up Transmute Calibration*

Description

Set Up Transmute Calibration

Usage

```
set.up.transmute.calibration(
  transmute.code,
  from.calibration.code,
  location,
  n.sim,
  n.chunks,
  allow.overwrite.cache = F,
  root.dir = get.jheem.root.directory(),
  return.simulations = F,
  verbose = T
)
```

---

simplot *Plot Simulations And Data*

Description

Plot Simulations And Data

Usage

```
simplot(
  ...,
  outcomes = NULL,
  corresponding.data.outcomes = NULL,
  split.by = NULL,
  facet.by = NULL,
  dimension.values = list(),
  dimension.values.post.mapping = list(),
  target.ontology = NULL,
  plot.which = c("sim.and.data", "sim.only")[1],
  summary.type = c("individual.simulation", "mean.and.interval",
    "median.and.interval")[1],
  omit.data.years = NULL,
  data.na.rm = NULL,
  data.locations = NULL,
  label.function = NULL,
  plot.year.lag.ratio = F,
  title = "location",
  title.suffix = NULL,
  n.facet.rows = NULL,
  append.url = F,
  data.manager = get.default.data.manager(),
  style.manager = get.default.style.manager(),
  show.data.pull.error = F,
```

```
  simset.names = NULL,
  debug = F
)
```

Arguments

`...`	One or more jheem.simulation.set objects and at most one character vector of outcomes (as an alternative to the 'outcomes' argument)
`outcomes`	A character vector of which simulation outcomes to plot
`corresponding.data.outcomes`	Specify directly which data outcomes should be plotted against simulation outcomes. Must be NULL or a character vector with outcomes as names; all of those outcomes must be present in either the 'outcomes' argument or in '...'"
`split.by`	At most one dimension
`facet.by`	Any number of dimensions but cannot include the split.by dimension
`dimension.values.post.mapping`	How to subset data in the form it will take after mappings have been applied to align sim and data ontologies. A list.
`plot.which`	Should simulation data and calibration data be plotted ('sim.and.data'), or only simulation data ('sim.only')
`omit.data.years`	Years to omit data, for example if data is available but wasn't calibrated to, while still showing simulation values for those years
`data.na.rm`	Sets na.rm = T or F during the data pull step. If NULL, will use T for rates, proportions, and ratios, and F for all other scales.
`data.locations`	Specifies locations for which to pull data. May be 1. NULL, in which case data locations will be inferred from the simsets' likelihoods, 2. a character vector of locations, or 3. a list of character vectors with outcomes as list names. For 3., outcomes not included will have their data locations inferred as in 1.
`label.function`	A function to reformat labels. If NULL, will use the first provided simset's "get.labels" property, or will do no transformation if plotting data only.
`title`	NULL or a single, non-NA character value. If "location", the location of the first provided simset (if any) will be used for the title.
`title.suffix`	NULL or a single, non-NA character value. Something to append to the end of the title that would be generated automatically based on the 'title' argument. Note that this pastes with no space, so add the space in if you want it.
`data.manager`	The data.manager from which to draw real-world data for the plots
`style.manager`	We are going to have to define this down the road. It's going to govern how we do lines and sizes and colors. For now, just hard code those in, and we'll circle back to it
`show.data.pull.error`	Not finding data does not block the plot from showing simulation projections, but if you'd like to see in error when data doesn't appear, set this to TRUE.
`simset.names`	A functional argument to help make sure the sims have names when they were plopped in by some other function that used them as variables.

Details

Returns a ggplot object: - With one panel for each combination of outcome x facet.by - x-axis is year - y-axis is outcome

---

`simplot.data.only` *Simplot Data Only*

Description

Simplot Data Only

Usage

```
simplot.data.only(
  outcomes,
  locations,
  split.by = NULL,
  facet.by = NULL,
  dimension.values = list(),
  dimension.values.post.mapping = list(),
  target.ontology = NULL,
  data.na.rm = NULL,
  label.function = NULL,
  plot.year.lag.ratio = F,
  title = "location",
  title.suffix = NULL,
  n.facet.rows = NULL,
  append.url = F,
  data.manager = get.default.data.manager(),
  style.manager = get.default.style.manager(),
  show.data.pull.error = F,
  debug = F
)
```

Arguments

`outcomes`	A character vector of which simulation outcomes to plot
`locations`	A character vector of locations for which to pull data.
`split.by`	At most one dimension
`facet.by`	Any number of dimensions but cannot include the split.by dimension
`dimension.values.post.mapping`	How to subset data in the form it will take after mappings have been applied to align sim and data ontologies. A list.
`data.na.rm`	Sets na.rm = T or F during the data pull step. If NULL, will use T for rates, proportions, and ratios, and F for all other scales.

`label.function` A function to reformat labels. If NULL, will use the first provided simset's "get.labels" property, or will do no transformation if plotting data only.

`title` NULL or a single, non-NA character value. If "location", the location of the first provided simset (if any) will be used for the title.

`title.suffix` NULL or a single, non-NA character value. Something to append to the end of the title that would be generated automatically based on the 'title' argument. Note that this pastes with no space, so add the space in if you want it.

`data.manager` The data.manager from which to draw real-world data for the plots

`style.manager` We are going to have to define this down the road. It's going to govern how we do lines and sizes and colors. For now, just hard code those in, and we'll circle back to it

`show.data.pull.error`
Not finding data does not block the plot from showing simulation projections, but if you'd like to see in error when data doesn't appear, set this to TRUE.

`standardize.age.strata.names`
Convert Age Strata Names to a Standard Form

Description

Convert Age Strata Names to a Standard Form

Usage

```
standardize.age.strata.names(strata.names)
```

Arguments

`strata.names` A character vector of names of age strata

Details

Calls `parse.age.strata.names` and feeds the output into `make.age.strata.names`

STATIC.FUNCTIONAL.FORM

Create an Static (Time-Invariant) Functional Form

Description

Create an Static (Time-Invariant) Functional Form

Create an Static (Time-Invariant) Functional Form

Super class

`jheem2::functional.form` -> `static.functional.form`

Methods

Public methods:

- `STATIC.FUNCTIONAL.FORM$new()`
- `STATIC.FUNCTIONAL.FORM$clone()`

Method `new()`:

Usage:

```
STATIC.FUNCTIONAL.FORM$new(
  value,
  link = c("identity", "log", "logistic")[1],
  min = NA,
  max = NA,
  value.name = "value",
  value.is.on.transformed.scale = F,
  overwrite.parameters.with.alphas = F,
  error.prefix = "Cannot create static functional.form: "
)
```

Arguments:

`value` array representing the value

`link` The name of the transformation to the scale on which alphas are added to the value. One of "identity", "log", "logistic"

`min, max` The minimum and maximum values this functional form can create. The default (NA) sets to the min/max for the link specified (eg, 0-Inf for a log link, 0-1 for a logistic link)

`value.is.on.transformed.scale` Logical indicating whether the the value is already on the transformed scale at which alphas apply

`overwrite.parameters.with.alphas` Logical indicating whether, when alphas are applied, they OVERWRITE values. If F, alphas are added to the transformed value (on the transformed scale)

Method `clone()`:

The objects of this class are cloneable with this method.

Usage:

```
STATIC.FUNCTIONAL.FORM$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`subset.simset` *Subset a Simulation Set*

Description

Subset a Simulation Set

Usage

```
## S3 method for class 'simset'
subset(simset, simulation.indices)
```

Arguments

`simulation.indices`
: An integer or logical vector specifying which simulations to keep.

`subset_data` *Subset a JHEEM Data Manager*

Description

Subset a JHEEM Data Manager

Usage

```
subset_data(
  data.manager = get.default.data.manager(),
  outcomes.to.keep = NULL,
  sources.to.keep = NULL,
  ontologies.to.keep = NULL,
  incomplete.dimension.values.to.keep = NULL,
  retain.ontology.registrations = T
)
```

Arguments

outcomes.to.keep, sources.to.keep, ontologies.to.keep	Character vectors of which outcomes, sources, and/or ontologies to retain data for.
incomplete.dimension.values.to.keep	A list of which years, locations, etc. to retain data for. Must be character vectors with the dimensions as names.
retain.ontology.registrations	Should registered ontologies be un-registered if no data remains that uses them?
A	jheem.data.manager object

---

thin.simset *Thin A Simulation Set*

Description

Subset a simulation set either by keeping every Nth simulation or keeping a fraction of all simulations. One of 'n' or 'keep' can be specified, with the other left as NULL.

Usage

```
thin.simset(simset, n = NULL, keep = NULL, allow.expand = F)
```

Arguments

n	Every Nth simulation will be kept, counting backwards from the last sim (ie., if n=3, the last simulation and the fourth-to-last simulation will be kept, etc.).
allow.expand	Whether to duplicate simulations if keep > simset$n.sim
k	Either a number of simulations to keep or a fraction to keep (between 0 and 1). Simulations will be drawn uniformly from the set.

Details

Decimal values above 1 are rounded down to the nearest whole number.

track.cumulative.outcome *Track a Combination of Cumulative Outcomes*

Description

Track a Combination of Cumulative Outcomes

Usage

```
track.cumulative.outcome(
  specification,
  name,
  outcome.metadata,
  value,
  value.is.numerator = F,
  denominator.outcome = NULL,
  corresponding.data.outcome = NULL,
  keep.dimensions = NULL,
  exclude.dimensions = NULL,
  subset.dimension.values = NULL,
  rename.dimension.values = NULL,
  force.dim.names.to.keep.dimensions = F,
  allow.expand.dimensions = NULL,
  allow.expand.denominator.dimensions = NULL,
  dimension.aliases = NULL,
  dimension.alias.suffix = NULL,
  scale = NULL,
  from.year = -Inf,
  to.year = Inf,
  save = T
)
```

Arguments

specification	The jheem.specification object to modify
name	The name by which to access the outcome value after the simulation runs. Cannot be 'infected', 'uninfected' or the name of another outcome
outcome.metadata	An object created by create.outcome.metadata, giving information on the outcome
value	Either (1) a single character value referencing a separately-registered cumulative outcome or a static (value does not change over time) quantity, or (2) an expression that includes only the names of cumulative outcomes and static model quantities and operators +, -, *, /, ^, sqrt, log, and exp

`value.is.numerator`
: In the case where the outcome's scale is proportion, rate, or ratio (any scale that results from a numerator or a denominator), an indicator of whether the value represents just the numerator or the actual proportion/rate/ratio

`denominator.outcome`
: A denominator for aggregating the outcome or calculating proportions. Required if the scale of this outcome is not 'number' or 'non.negative.number', otherwise optional. Must be the name of a separately registered cumulative outcome that has type 'non.negative.number'

`corresponding.data.outcome`
: The (optional) name of a real-world outcome (with observations registered to a data.manager) to which this model outcome corresponds

`keep.dimensions`
: The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`exclude.dimensions`
: Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`subset.dimension.values`
: A list of dimension values indicating for which values of which dimensions the outcome should be tracked

`rename.dimension.values`
: A list of names character vectors indicating which dimension values should be renamed in the outcome ontology

`force.dim.names.to.keep.dimensions`
: A logical indicating whether the dim.names of the outcome should be forced to match keep.dimensions (based on dimension values in compartments.for.infected.only, compartments.for.uninfected.only, and compartments.for.infected.and.uninfected) regardless of the dimensions of the underlying model quantities that it comprises.

`allow.expand.dimensions`, `allow.expand.denominator.dimensions`
: A character vector of dimensions which can be added to the outcome (or outcome's denominator) even if not present in its component outcomes or quantities. The outcome will have the same value for all values in that dimension

`dimension.aliases`
: A named character vector indicating what dimensions in the inputs to the outcome (the names of the vector) should be converted to (the values of the vector)

`dimension.alias.suffix`
: A single character value, indicating a suffix that should be removed from the dimensions of the inputs to the outcome. Eg, if ".to" is specified, then dimension "race.to" in outcome inputs becomes "race" in the outcome

`scale`
: The scale of this outcome. Can be NULL when outcome.metadata is not NULL (only required when not saving and not using outcome.metadata)

`from.year`, `to.year`
: The time span during which the outcome should be recorded

save — A logical indicator of whether this outcome should be stored in simulations. If FALSE, the outcome will be available for calculating other outcomes (with track.cumulative.outcome or track.point.outcome), but will not be retrievable afterwards

Details

Creates a cumulative outcome to track by combining other cumulative outcomes and static quantities

---

track.cumulative.proportion.from.rate

Track an Outcome Formed by Integrating a Time-Varying Rate into a Proportion

---

Description

Track an Outcome Formed by Integrating a Time-Varying Rate into a Proportion

Usage

```
track.cumulative.proportion.from.rate(
  specification,
  name,
  outcome.metadata,
  rate.value,
  denominator.outcome,
  corresponding.data.outcome = NULL,
  calculate.proportion.leaving = T,
  keep.dimensions = NULL,
  exclude.dimensions = NULL,
  subset.dimension.values = NULL,
  rename.dimension.values = NULL,
  force.dim.names.to.keep.dimensions = F,
  allow.expand.dimensions = NULL,
  allow.expand.denominator.dimensions = NULL,
  dimension.aliases = NULL,
  dimension.alias.suffix = NULL,
  from.year = -Inf,
  to.year = Inf,
  save = T
)
```

Arguments

specification — The jheem.specification object to modify

`name` The name by which to access the outcome value after the simulation runs. Cannot be 'infected', 'uninfected' or the name of another outcome

`outcome.metadata`
An object created by `create.outcome.metadata`, giving information on the outcome

`rate.value` The rate to be integrated. Either (1) a single character value referencing a separately-registered "point" (non-cumulative) outcome or a quantity, or (2) an expression that includes only the names of point outcomes and model quantities and operators +, -, *, /, ^, sqrt, log, and exp

`denominator.outcome`
A denominator for aggregating the outcome or calculating proportions. Required if the scale of this outcome is not 'number' or 'non.negative.number', otherwise optional. Must be the name of a separately registered cumulative outcome that has type 'non.negative.number'

`corresponding.data.outcome`
The (optional) name of a real-world outcome (with observations registered to a data.manager) to which this model outcome corresponds

`calculate.proportion.leaving`
If TRUE, integrates to get the proportion of those who "leave" the imaginary compartment at the given rate. If FALSE, calculates the proportion who have not left (ie 1-proportion.leaving)

`keep.dimensions`
The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`exclude.dimensions`
Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`subset.dimension.values`
A list of dimension values indicating for which values of which dimensions the outcome should be tracked

`rename.dimension.values`
A list of names character vectors indicating which dimension values should be renamed in the outcome ontology

`force.dim.names.to.keep.dimensions`
A logical indicating whether the dim.names of the outcome should be forced to match keep.dimensions (based on dimension values in compartments.for.infected.only, compartments.for.uninfected.only, and compartments.for.infected.and.uninfected) regardless of the dimensions of the underlying model quantities that it comprises.

`allow.expand.dimensions`, `allow.expand.denominator.dimensions`
A character vector of dimensions which can be added to the outcome (or outcome's denominator) even if not present in its component outcomes or quantities. The outcome will have the same value for all values in that dimension

`dimension.aliases`
: A named character vector indicating what dimensions in the inputs to the outcome (the names of the vector) should be converted to (the values of the vector)

`dimension.alias.suffix`
: A single character value, indicating a suffix that should be removed from the dimensions of the inputs to the outcome. Eg, if ".to" is specified, then dimension "race.to" in outcome inputs becomes "race" in the outcome

`from.year, to.year`
: The time span during which the outcome should be recorded

`save`
: A logical indicator of whether this outcome should be stored in simulations. If FALSE, the outcome will be available for calculating other outcomes (with `track.cumulative.outcome` or `track.point.outcome`), but will not be retrievable afterwards

Details

Calculates a proportion derived from having a 'compartment' which people leave at a given rate, and calculates the proportion who either leave or remain in that compartment over time

---

`track.dynamic.outcome` *Mark a Transition for Tracking*

Description

Mark a Transition for Tracking

Usage

```
track.dynamic.outcome(
  specification,
  name,
  outcome.metadata,
  dynamic.quantity.name,
  denominator.outcome = NULL,
  include.tags = NULL,
  exclude.tags = NULL,
  groups = NULL,
  multiply.by = NULL,
  corresponding.data.outcome = NULL,
  keep.dimensions = NULL,
  exclude.dimensions = NULL,
  subset.dimension.values = NULL,
  rename.dimension.values = NULL,
  allow.expand.dimensions = NULL,
  allow.expand.denominator.dimensions = NULL,
  dimension.aliases = NULL,
  dimension.alias.suffix = NULL,
```

```
  scale = NULL,
  from.year = -Inf,
  to.year = Inf,
  save = T
)
```

Arguments

specification	The jheem.specification object to modify
name	The name by which to access the outcome value after the simulation runs. Cannot be 'infected', 'uninfected' or the name of another outcome
outcome.metadata	An object created by create.outcome.metadata, giving information on the outcome
dynamic.quantity.name	The name of the dynamic quantity to track for the outcome. Must be one of 'population', 'mortality', 'births', 'births.from', 'births.to', 'incidence', 'incidence.to', 'incidence.from', 'incidence.by', 'remission', 'remission.from', 'remission.to'
denominator.outcome	Optional: A denominator for calculating proportions. Must be the name of a separately registered cumulative outcome that has type 'non.negative.number'
include.tags	The tags to track for (as given in the call to register.transition). Choosing NULL (the default value) tracks for all tags for which the outcome could apply minus excluded tags
exclude.tags	Tags to exclude from tracking.
groups	The groups to track for ('infected' or 'uninfected'). Choosing NULL (the default value) tracks for all groups for which the outcome could apply
multiply.by	An optional quantity to multiply the outcome value by at each time step before storing. Can be either (1) a numeric value, (2) a single character value referencing a model quantity or element, or (3) an expression that includes only the names of model quantities and elements and operators +, -, *, /, ^, sqrt, log, and exp
corresponding.data.outcome	The (optional) name of a real-world outcome (with observations registered to a data.manager) to which this model outcome corresponds
keep.dimensions	The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.
exclude.dimensions	Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.
subset.dimension.values	A list of dimension values indicating for which values of which dimensions the outcome should be tracked

`rename.dimension.values`
: A list of names character vectors indicating which dimension values should be renamed in the outcome ontology

`allow.expand.dimensions`, `allow.expand.denominator.dimensions`
: A character vector of dimensions which can be added to the outcome (or outcome's denominator) even if not present in its component outcomes or quantities. The outcome will have the same value for all values in that dimension

`dimension.aliases`
: A named character vector indicating what dimensions in the inputs to the outcome (the names of the vector) should be converted to (the values of the vector)

`dimension.alias.suffix`
: A single character value, indicating a suffix that should be removed from the dimensions of the inputs to the outcome. Eg, if ".to" is specified, then dimension "race.to" in outcome inputs becomes "race" in the outcome

`scale`
: The scale of this outcome. Can be NULL when outcome.metadata is not NULL (only required when not saving and not using outcome.metadata)

`from.year`, `to.year`
: The time span during which the outcome should be recorded

`save`
: A logical indicator of whether this outcome should be stored in simulations. If FALSE, the outcome will be available for calculating other outcomes (with `track.cumulative.outcome` or `track.point.outcome`), but will not be retrievable afterwards

Details

Integrates the dynamic quantity (or the product of the dynamic quantity x multiply by), such that the simulation stores, for each year y, the integral from y to y+1 of (dynamic quantity * multiply.by)

`track.integrated.outcome`
Mark an Outcome to Be Integrated and Tracked

Description

Mark an Outcome to Be Integrated and Tracked

Usage

```
track.integrated.outcome(
  specification,
  name,
  outcome.metadata,
  value.to.integrate,
  value.is.numerator = F,
  multiply.by = NULL,
  denominator.outcome = NULL,
```

```
  corresponding.data.outcome = NULL,
  keep.dimensions = NULL,
  exclude.dimensions = NULL,
  subset.dimension.values = NULL,
  rename.dimension.values = NULL,
  force.dim.names.to.keep.dimensions = F,
  allow.expand.dimensions = NULL,
  allow.expand.denominator.dimensions = NULL,
  dimension.aliases = NULL,
  dimension.alias.suffix = NULL,
  scale = NULL,
  from.year = -Inf,
  to.year = Inf,
  save = T
)
```

Arguments

specification	The jheem.specification object to modify
name	The name by which to access the outcome value after the simulation runs. Cannot be 'infected', 'uninfected' or the name of another outcome
outcome.metadata	An object created by create.outcome.metadata, giving information on the outcome
value.to.integrate	Either (1) a single character value referencing a separately-registered "point" (non-cumulative) outcome or a quantity, or (2) an expression that includes only the names of point outcomes and model quantities and operators +, -, *, /, ^, sqrt, log, and exp
value.is.numerator	In the case where the outcome's scale is proportion, rate, or ratio (any scale that results from a numerator or a denominator), an indicator of whether the value.to.integrate represents just the numerator or the actual proportion/rate/ratio
multiply.by	An optional quantity to multiply the outcome value by at each time step before storing. Can be either (1) a numeric value, (2) a single character value referencing a model quantity or element, or (3) an expression that includes only the names of model quantities and elements and operators +, -, *, /, ^, sqrt, log, and exp
denominator.outcome	Optional: A denominator for calculating proportions. Must be the name of a separately registered cumulative outcome that has type 'non.negative.number'
corresponding.data.outcome	The (optional) name of a real-world outcome (with observations registered to a data.manager) to which this model outcome corresponds
keep.dimensions	The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

exclude.dimensions
: Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

subset.dimension.values
: A list of dimension values indicating for which values of which dimensions the outcome should be tracked

rename.dimension.values
: A list of names character vectors indicating which dimension values should be renamed in the outcome ontology

force.dim.names.to.keep.dimensions
: A logical indicating whether the dim.names of the outcome should be forced to match keep.dimensions (based on dimension values in compartments.for.infected.only, compartments.for.uninfected.only, and compartments.for.infected.and.uninfected) regardless of the dimensions of the underlying model quantities that it comprises.

allow.expand.dimensions, allow.expand.denominator.dimensions
: A character vector of dimensions which can be added to the outcome (or outcome's denominator) even if not present in its component outcomes or quantities. The outcome will have the same value for all values in that dimension

dimension.aliases
: A named character vector indicating what dimensions in the inputs to the outcome (the names of the vector) should be converted to (the values of the vector)

dimension.alias.suffix
: A single character value, indicating a suffix that should be removed from the dimensions of the inputs to the outcome. Eg, if ".to" is specified, then dimension "race.to" in outcome inputs becomes "race" in the outcome

scale
: The scale of this outcome. Can be NULL when outcome.metadata is not NULL (only required when not saving and not using outcome.metadata)

from.year, to.year
: The time span during which the outcome should be recorded

save
: A logical indicator of whether this outcome should be stored in simulations. If FALSE, the outcome will be available for calculating other outcomes (with track.cumulative.outcome or track.point.outcome), but will not be retrievable afterwards

Details

Tracking outcomes with this function allows "point" outcomes to be converted into "cumulative" outcomes without going through the differential equation solver. This function yields an approximation of the outcome that would be generated by using track.dynamic.outcome in a way that is computationally much less expensive when running simulations

`track.point.outcome` *Track a Combination of "Point" (Non-Cumulative) Outcomes*

Description

Track a Combination of "Point" (Non-Cumulative) Outcomes

Usage

```
track.point.outcome(
  specification,
  name,
  outcome.metadata,
  value,
  value.is.numerator = F,
  denominator.outcome = NULL,
  corresponding.data.outcome = NULL,
  keep.dimensions = NULL,
  exclude.dimensions = NULL,
  subset.dimension.values = NULL,
  rename.dimension.values = NULL,
  force.dim.names.to.keep.dimensions = F,
  allow.expand.dimensions = NULL,
  allow.expand.denominator.dimensions = NULL,
  dimension.aliases = NULL,
  dimension.alias.suffix = NULL,
  scale = NULL,
  from.year = -Inf,
  to.year = Inf,
  save = T
)
```

Arguments

`specification`	The jheem.specification object to modify
`name`	The name by which to access the outcome value after the simulation runs. Cannot be 'infected', 'uninfected' or the name of another outcome
`outcome.metadata`	An object created by `create.outcome.metadata`, giving information on the outcome
`value`	Either (1) a single character value referencing a separately-registered "point" (non-cumulative) outcome or a quantity, or (2) an expression that includes only the names of point outcomes and model quantities and operators +, -, *, /, ^, sqrt, log, and exp

`value.is.numerator`
: In the case where the outcome's scale is proportion, rate, or ratio (any scale that results from a numerator or a denominator), an indicator of whether the value represents just the numerator or the actual proportion/rate/ratio

`denominator.outcome`
: A denominator for aggregating the outcome or calculating proportions. Required if the scale of this outcome is not 'number' or 'non.negative.number', otherwise optional. Must be the name of a separately registered "point" (non-cumulative) outcome that has type 'non.negative.number'

`corresponding.data.outcome`
: The (optional) name of a real-world outcome (with observations registered to a data.manager) to which this model outcome corresponds

`keep.dimensions`
: The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`exclude.dimensions`
: Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`subset.dimension.values`
: A list of dimension values indicating for which values of which dimensions the outcome should be tracked

`rename.dimension.values`
: A list of names character vectors indicating which dimension values should be renamed in the outcome ontology

`force.dim.names.to.keep.dimensions`
: A logical indicating whether the dim.names of the outcome should be forced to match keep.dimensions (based on dimension values in compartments.for.infected.only, compartments.for.uninfected.only, and compartments.for.infected.and.uninfected) regardless of the dimensions of the underlying model quantities that it comprises.

`allow.expand.dimensions`, `allow.expand.denominator.dimensions`
: A character vector of dimensions which can be added to the outcome (or outcome's denominator) even if not present in its component outcomes or quantities. The outcome will have the same value for all values in that dimension

`dimension.aliases`
: A named character vector indicating what dimensions in the inputs to the outcome (the names of the vector) should be converted to (the values of the vector)

`dimension.alias.suffix`
: A single character value, indicating a suffix that should be removed from the dimensions of the inputs to the outcome. Eg, if ".to" is specified, then dimension "race.to" in outcome inputs becomes "race" in the outcome

`scale`
: The scale of this outcome. Can be NULL when outcome.metadata is not NULL (only required when not saving and not using outcome.metadata)

`from.year`, `to.year`
: The time span during which the outcome should be recorded

save — A logical indicator of whether this outcome should be stored in simulations. If FALSE, the outcome will be available for calculating other outcomes (with track.cumulative.outcome or track.point.outcome), but will not be retrievable afterwards

Details

Creates a "point" (non-cumulative) outcome to track by combining other point outcomes and quantities

---

track.sub.version.outcomes

Track Sub Version Outcomes for a Specification

Description

Track Sub Version Outcomes for a Specification

Usage

```
track.sub.version.outcomes(
  specification,
  sub.versions,
  outcome.names,
  keep.dimensions = NULL,
  subset.dimension.values = NULL,
  ontology.mapping = NULL,
  from.year = NULL,
  to.year = NULL
)
```

---

track.transition

Mark a Transition for Tracking

Description

Mark a Transition for Tracking

Usage

```
track.transition(
  specification,
  name,
  outcome.metadata,
  dimension,
  from.compartments,
  to.compartments,
  denominator.outcome = NULL,
  include.tags = NULL,
  exclude.tags = NULL,
  groups = NULL,
  multiply.by = NULL,
  corresponding.data.outcome = NULL,
  keep.dimensions = NULL,
  exclude.dimensions = NULL,
  subset.dimension.values = NULL,
  rename.dimension.values = NULL,
  allow.expand.dimensions = NULL,
  allow.expand.denominator.dimensions = NULL,
  dimension.aliases = NULL,
  dimension.alias.suffix = NULL,
  scale = NULL,
  from.year = -Inf,
  to.year = Inf,
  save = T
)
```

Arguments

specification	The jheem.specification object to modify
name	The name by which to access the outcome value after the simulation runs. Cannot be 'infected', 'uninfected' or the name of another outcome
outcome.metadata	An object created by create.outcome.metadata, giving information on the outcome
dimension	The name of the (single) dimension along which we are tracking a transition
from.compartments, to.compartments	The names (or aliases) of the from and to compartments of the transition to track that gives information about the resulting quantity
denominator.outcome	Optional: A denominator for calculating proportions. Must be the name of a separately registered cumulative outcome that has type 'non.negative.number'
include.tags	The tags to track for (as given in the call to register.transition). Choosing NULL (the default value) tracks for all tags for which the outcome could apply minus excluded tags
exclude.tags	Tags to exclude from tracking.

`groups` The groups to track for ('infected' or 'uninfected'). Choosing NULL (the default value) tracks for all groups for which the outcome could apply

`multiply.by` An optional quantity to multiply the outcome value by at each time step before storing. Can be either (1) a numeric value, (2) a single character value referencing a model quantity or element, or (3) an expression that includes only the names of model quantities and elements and operators +, -, *, /, ^, sqrt, log, and exp

`corresponding.data.outcome`
The (optional) name of a real-world outcome (with observations registered to a data.manager) to which this model outcome corresponds

`keep.dimensions`
The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`exclude.dimensions`
Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

`subset.dimension.values`
A list of dimension values indicating for which values of which dimensions the outcome should be tracked

`rename.dimension.values`
A list of names character vectors indicating which dimension values should be renamed in the outcome ontology

`allow.expand.dimensions`, `allow.expand.denominator.dimensions`
A character vector of dimensions which can be added to the outcome (or outcome's denominator) even if not present in its component outcomes or quantities. The outcome will have the same value for all values in that dimension

`dimension.aliases`
A named character vector indicating what dimensions in the inputs to the outcome (the names of the vector) should be converted to (the values of the vector)

`dimension.alias.suffix`
A single character value, indicating a suffix that should be removed from the dimensions of the inputs to the outcome. Eg, if ".to" is specified, then dimension "race.to" in outcome inputs becomes "race" in the outcome

`scale` The scale of this outcome. Can be NULL when outcome.metadata is not NULL (only required when not saving and not using outcome.metadata)

`from.year`, `to.year`
The time span during which the outcome should be recorded

`save` A logical indicator of whether this outcome should be stored in simulations. If FALSE, the outcome will be available for calculating other outcomes (with `track.cumulative.outcome` or `track.point.outcome`), but will not be retrievable afterwards

Details

Integrates the transition (or the product of transition x multiply by), such that the simulation stores, for each year y, the integral from y to y+1 of (transition * multiply.by)

union.dimension.values

Get the union of dimension values sets

Description

Get the union of dimension values sets

Usage

```
union.dimension.values(..., error.prefix = "")
```

Arguments

...	can be either (1) dimension values or (2) lists of dimension values

union.target.populations

Create a Target Population that is the UNION Other Target Populations

Description

Create a Target Population that is the UNION Other Target Populations

Usage

```
union.target.populations(..., name = NULL)
```

Arguments

...	One or more target.population objects or lists of target populations
name	A short (<= 20 characters) descriptive name for the target population. Should be nicely formatted for public display

See Also

Other Functions to create Target Populations: `create.target.population()`, `diff.target.populations()`, `intersect.target.populations()`

update.outcome.keep.dimensions

Change the keep.dimensions of an outcome registered to an ancestor specification

Description

Change the keep.dimensions of an outcome registered to an ancestor specification

Usage

```
## S3 method for class 'outcome.keep.dimensions'
update(specification, outcome.name, keep.dimensions, exclude.dimensions = NULL)
```

Arguments

specification	The jheem.specification object to modify
outcome.name	The name of the outcome to update keep.dimensions for
keep.dimensions	The dimensions which to keep in storing outcome values. All other dimensions are marginalized. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.
exclude.dimensions	Dimensions to be marginalized out when storing outcome values. Either keep.dimensions OR exclude.dimensions but not both can be set; if both are NULL, will keep all possible dimensions for the outcome.

using.data.binding *Evaluate an expression under a data binding*

Description

Makes the binding active while 'expr' is evaluated, and restores the previous active binding afterwards, also on error.

Usage

```
using.data.binding(binding, expr)
```

Arguments

binding	A 'jheem.data.binding' object
expr	The expression to evaluate

Value

The value of 'expr'

`version.has.been.registered` *Check Whether a JHEEM Model Specification has been Registered for a Version*

Description

Check Whether a JHEEM Model Specification has been Registered for a Version

Usage

```
version.has.been.registered(version)
```

Arguments

`version`	A character vector with one or more version names

Value

A logical vector of the same length as 'version'

See Also

Other JHEEM Version Management Functions: `get.prior.versions()`, `get.specification.for.version()`, `register.model.specification()`

Index

∗ **Basic Likelihoods**
create.basic.likelihood.instructions, 24
create.basic.likelihood.instructions.with.included.multiplier, 26
create.basic.likelihood.instructions.with.specified.outcome, 29
create.time.lagged.comparison.likelihood.instructions, 70
create.time.lagged.comparison.likelihood.instructions.with.included.multiplier, 72
∗ **Defining Core Components for a Model Specification**
register.aging, 142
register.model.mechanism, 153
register.mortality, 156
register.natality, 157
register.remission, 160
register.transition, 162
register.transmission, 163
∗ **Functions to create Target Populations**
create.target.population, 69
diff.target.populations, 79
intersect.target.populations, 111
union.target.populations, 203
∗ **Functions to create and modify a model settings**
set.element.functional.form.from.time, 174
set.element.functional.form.future.slope, 175
set.element.functional.form.interaction.alphas, 176
set.element.ramp.times, 178
∗ **Functions to create and modify model settings**
set.element.functional.form.main.effect.alphas, 177
∗ **Functions to identify why ontologies fail to map**
apply.ontology.mapping, 10
get.identity.ontology.mapping, 95
identity.ontology, 124
∗ **Functions to modify model settings**
set.element.value, 180
∗ **JHEEM Version Management Functions**
get.prior.versions, 101
get.specification.for.version, 106
register.model.specification, 156
version.has.been.registered, 205
∗ **Nested Proportion Likelihoods**
create.nested.proportion.likelihood.instructions, 55
create.nested.proportion.likelihood.instructions.with.included.multiplier, 59
create.time.lagged.comparison.nested.proportion.likelihood.instructions, 75
get.p.bias.estimates, 100
[.ontology, 7
[<-.ontology, 8
[[<-.ontology, 7
$<-.ontology, 8

add.sub.version, 9
anchor.intervention.effect.location.and.version, 9
apply.ontology.mapping, 10, *95*, *125*
are.simulations.present, 11
array.access, 11
as.list.ontology, 12
as.ontology, 13
assemble.mcmc.from.calibration, 13
assemble.simulations.from.calibration, 14
assemble.transmuted.simulations, 14

burn.simset, 15

cache.mcmc.summary, 15

can.transmute.versions, 16
clear.all.managers, 16
clear.calibration.cache, 17
clear.calibrations, 17
clear.interventions, 17
clear.ontology.error.debug.info, 18
clear.ontology.mappings, 18
clear.transmute.calibration.cache, 18
clear.versions, 19
collapse.array.according.to.indices, 19
combine.ontology.mappings, 20
compare.sims, 20
compute.likelihood, 21
compute.likelihood(), *22*
compute.likelihood.piecewise, 21
copy.calibration.info, 22
copy.data.manager, 23
copy.simulation.set, 23
create.basic.likelihood.instructions, 24, *29*, *31*, *72*, *75*
create.basic.likelihood.instructions.with.included.multiplier, 26, *26*, *31*, *72*, *75*
create.basic.likelihood.instructions.with.specified.outcome, *26*, 29, *29*, *72*, *75*
create.basic.ratio.likelihood.instructions, 31
create.bernoulli.likelihood.instructions, 33
create.custom.likelihood.instructions, 34
create.data.binding, 34
create.data.manager, 35
create.data.manager.alias, 35
create.ifelse.likelihood.instructions, 36
create.intervention, 36, *52*
create.intervention.criterion, *52*
create.intervention.effect, 37
create.jheem.engine, 38
create.jheem.specification, 39, *156*
create.jheem.transmuter, 41
create.likelihood.weights, *25*, *28*, *30*, *33*, *34*, 42, *58*, *62*, *71*, *74*, *78*
create.linear.functional.form, 42
create.linear.spline.functional.form, 44
create.location.based.ifelse.likelihood.instructions, 45
create.log.linear.functional.form, 46
create.logistic.linear.functional.form, 47
create.logistic.spline.functional.form, 48
create.logistic.tail.functional.form, 49
create.model.foreground, 51, *152*
create.monotonic.criteria.based.intervention, 51
create.monotonic.criterion, 52
create.natural.spline.functional.form, 54
create.nested.proportion.likelihood.instructions, 55, *63*, *79*, *101*
create.nested.proportion.likelihood.instructions.with.incl 59, *59*, *79*, *101*
create.ontology.mapping, 63
create.other.catchall.ontology.mapping, 64
create.outcome.location.mapping, 65
create.outcome.metadata, 65, *147*, *189*, *192*, *194*, *196*, *198*, *201*
create.simset.collection, 66
create.solver.metadata, *39*, 67, *171*
create.static.functional.form, 67
create.style.manager, 68
create.target.population, 69, *80*, *111*, *203*
create.time.lagged.comparison.likelihood.instructions, *26*, *29*, *31*, 70, *75*
create.time.lagged.comparison.likelihood.instructions.with *26*, *29*, *31*, 72, *72*
create.time.lagged.comparison.nested.proportion.likelihood *59*, *63*, 75, *101*

declare.data.managers, 79
diff.target.populations, *70*, 79, *111*, *203*
disable.jheem.solver.tracking, 80
distribute.dimension.values, 80

enable.jheem.solver.tracking, 81
examine.get.mappings.to.align.ontologies, 81
examine.get.ontology.mapping, 82
examine.recent.failed.ontology.mapping, 83
extract.plotly.plot, 83

execute.simplot, 84
expand.array, 85
extract.last.simulation.from.calibration, 85

fast.array.access *(array.access)*, 11
find.ontology.mapping, 86

get.calibration.progress, 87
get.collapse.array.indices, 88
get.cumulative.age.counts.smoother, 88
get.data.binding, 89, *91*
get.data.binding.identity, 89
get.data.inputs, 90, *141*
get.data.manager, 91, *110*
get.data.manager.provenance, *89*, 91, *123*
get.data.outcome.descriptions *(get.data.outcome.pretty.names)*, 92
get.data.outcome.labels *(get.data.outcome.pretty.names)*, 92
get.data.outcome.pretty.names, 92
get.data.source.full.names, 93
get.data.source.short.names *(get.data.source.full.names)*, 93
get.default.data.manager, 93
get.default.style.manager, 94
get.expand.array.indices, 94
get.expand.then.collapse.array.indices, 94
get.identity.ontology.mapping, *10*, 95, *125*
get.jheem.root.directory, 95
get.jheem.solver.tracked.info, 96
get.locations.with.data, 96
get.mappings.to.align.ontologies, *10*, *81*, 97, *99*
get.mcmc.mixing.statistic, 97
get.ontology.error.debug.info, 98
get.ontology.mapping, *10*, *82*, 98, *99*
get.ontology.mapping.matrix, 99
get.p.bias.estimates, *59*, *63*, *79*, 100
get.prior.versions, 101, *106*, *156*, *205*
get.registered.ontology, 102
get.simset.data, 102
get.simset.filename, 103
get.simset.filename.and.dir, 104
get.simset.params, 104
get.simulation.metadata, 105
get.specification.for.version, *101*, 106, *156*, *205*
get.specification.metadata, 106
get.year.bounds.for.outcome, 107

import.data, 107
incomplete.dimensions, 108
incomplete.dimensions.ontology, 108
inspect.marginals, 109
instantiate.likelihood, 110
interpolate, 110
intersect.target.populations, *70*, *80*, 111, *203*
is.complete *(is_complete)*, 112
is.ontology, 111
is_complete, 112
is_complete.ontology, 112

JHEEM, 113
jheem2, 118
jheem2-calibration-params, 118
jheem2-data-manager-params, 119
jheem2-likelihood-params, 119
jheem2-package *(jheem2)*, 118
jheem2-params, 119
jheem2-simset-params, 120
join.interventions, *52*, 120
join.likelihood.instructions, 121
join.outcome.location.mappings, 121
join.simulation.sets, 122

list.data.managers, 122
load.data.manager, *90*, 123, *172*
load.simulation.set, 123

make.age.strata.names, 124, *131*, *185*
map.value.ontology, *10*, *95*, 124
mcmc.traceplot, 125

names<-.ontology, 125

ontology, *7*, *8*, *12*, *83*, *99*, *108*, *112*, 126, *136*, *167*
ontology(), *119*, *146*
ONTOLOGY.MAPPING, 126
outer.join.dim.names, 130

parse.age.strata.names, 131, *185*

parse.simset.filename, 131
parse.year.names, 131
percentage.cache.complete, 132
plot.data.validation, 132
plot.simulations, 133
prepare.plot, 134
print.ontology, 135
pull.data, 136
put.data, 138
put.data.long.form, 139

rebuild.data.binding, 141, *172*
register.aging, 142, *153*, *157, 158*, *161*, *163, 164*
register.calibrated.parameters.for.version, 143
register.calibration.info, 144
register.data.manager, *35*, 145, *172*
register.data.metadata.field, *139*, *141*, 145
register.data.ontology, *137*, *139, 140*, 146, *166*
register.data.outcome, *109*, *136*, *139, 140*, 146, *166*
register.data.source, *109*, *139, 140*, 147, *166*
register.default.parameter.values, 148
register.fixed.model.strata, 148
register.initial.population, 149
register.model.element, 149
register.model.element.values, 151
register.model.foreground, 152
register.model.mechanism, *142*, 153, *157, 158*, *161*, *163, 164*
register.model.quantity, 154
register.model.quantity.subset, 155
register.model.specification, *101*, *106*, *143*, 156, *161, 162*, *205*
register.mortality, *142*, *153*, 156, *158*, *161*, *163, 164*
register.natality, *142*, *153*, 157, *157*, *161*, *163, 164*
register.ontology.mapping, *10*, *99*, 158
register.parent.data.source, 159
register.remission, *142*, *153*, *157, 158*, 160, *163, 164*
register.sampled.parameters.for.version, 161
register.set.parameters.for.version, 162
register.transition, *142*, *153*, *157, 158*, *161*, 162, *164*, *194*, *201*
register.transmission, *142*, *153*, *157, 158*, *161*, 163, *163*
register.transmute.calibration.info, 165
remove.data, 165
rerun.simulations, 166
resolve.intervention.effect, 167
resolve.ontology.dimension.values, 167
restratify.age.counts, 168
retrieve.simulation.set, 169
run.calibration, 169
run.simulations.from.parameters, 170
run.transmute.calibration, 171

save.simulation.set, 171
set.data.binding, *34*, *89*, 172
set.data.release.resolver, *141*, 172
set.default.data.manager, 173
set.default.style.manager, 173
set.element.functional.form.alphas.from.parameters, 173
set.element.functional.form.from.time, 174, *176, 177*, *180*
set.element.functional.form.future.slope, 175, *175*, *177*, *180*
set.element.functional.form.interaction.alphas, *175*, 176, *176*, *180*
set.element.functional.form.main.effect.alphas, 177
set.element.functional.form.to.time *(*set.element.functional.form.from.time*)*, 174
set.element.ramp.times, *175–177*, 178
set.element.ramp.values *(*set.element.ramp.times*)*, 178
set.element.taper.times *(*set.element.ramp.times*)*, 178
set.element.taper.values *(*set.element.ramp.times*)*, 178
set.element.value, 180
set.jheem.root.directory, 180
set.up.calibration, 181
set.up.transmute.calibration, 181
simplot, 182
simplot.data.only, 184

standardize.age.strata.names, 185
STATIC.FUNCTIONAL.FORM, 186
subset.simset, 187
subset_data, 187

thin.simset, 188
track.cumulative.outcome, 189, *191*, *193*, *195*, *197*, *200*, *202*
track.cumulative.proportion.from.rate, 191
track.dynamic.outcome, 193, *197*
track.dynamic.quantity, *66*
track.integrated.outcome, 195
track.point.outcome, *191*, *193*, *195*, *197*, 198, *200*, *202*
track.quantity, *66*
track.sub.version.outcomes, 200
track.transition, *66*, 200

union.dimension.values, 203
union.target.populations, *70*, *80*, *111*, 203
update.outcome.keep.dimensions, 204
using.data.binding, *34*, *89*, 204

version.has.been.registered, *101*, *106*, *156*, 205