Package: madrat (via r-universe)

August 21, 2024

Type Package

Title May All Data be Reproducible and Transparent (MADRaT) *

Version 3.13.0

Date 2024-08-21

Description Provides a framework which should improve reproducibility and transparency in data processing. It provides functionality such as automatic meta data creation and management, rudimentary quality management, data caching, work-flow management and data aggregation. * The title is a wish not a promise. By no means we expect this package to deliver everything what is needed to achieve full reproducibility and transparency, but we believe that it supports efforts in this direction.

License BSD_2_clause + file LICENSE

URL https://github.com/pik-piam/madrat, https://doi.org/10.5281/zenodo.1115490

BugReports https://github.com/pik-piam/madrat/issues

Depends magclass (>= 5.7.0), R (>= 2.10.0)

Imports callr, digest, igraph (>= 2.0.0), Matrix, methods, pkgload, renv, stringi, tools, utils, withr, yaml

Suggests covr, ggplot2, graphics, grDevices, knitr, rmarkdown, terra, testthat, tibble

VignetteBuilder knitr

Config/Keywords tool

Encoding UTF-8

LazyData no

RoxygenNote 7.3.2

Repository https://pik-piam.r-universe.dev

RemoteUrl https://github.com/pik-piam/madrat

RemoteRef HEAD

RemoteSha ff3c9f8160d6ddc444002fd3bc17d7a0eccaba99

Contents

`madrat-package` *May All Data be Reproducible And Transparent (madrat) **

Description

Package provides a basic framework which should improve reproducibility and transparency in data processing. It provides functionality such as automatic meta data creation and management, rudimentary quality management, data caching, work-flow management an data aggregation.

Details

* The title is a wish not a promise. By no means we expect this package to deliver everything what is needed to achieve full reproducibility and transparency, but we believe that it supports efforts in this direction.

Author(s)

Maintainer: Jan Philipp Dietrich <dietrich@pik-potsdam.de> (ORCID) (Potsdam Institute for Climate Impact Research)

Authors:

- Lavinia Baumstark <lavinia@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)
- Stephen Wirth <wirth@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)
- Anastasis Giannousakis <giannou@pik-potsdam.de>
- Renato Rodrigues <Renato.Rodrigues@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)
- Benjamin Leon Bodirsky <bodirsky@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)
- Debbora Leip <leip@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)
- Ulrich Kreidenweis <kreidenweis@pik-potsdam.de>
- David Klein <dklein@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)
- Pascal Sauer <pascal.sauer@pik-potsdam.de> (Potsdam Institute for Climate Impact Research)

See Also

Useful links:

- https://github.com/pik-piam/madrat
- doi:10.5281/zenodo.1115490
- Report bugs at https://github.com/pik-piam/madrat/issues

`addMapping` *addMapping*

Description

Function whichs adds another mapping to the current list of extramappings in the madrat configuration (see `setConfig`) and stores the mapping in the mapping folder as well as output folder.

Usage

```
addMapping(filename, mapping = NULL)
```

Arguments

`filename`	The name of the the region mapping that should added including file ending (e.g. "regionmappingREMIND.csv"). Supported formats are currently ".csv" and ".rds".
`mapping`	Mapping provided as data.frame, or NULL. If a mapping is provided the data will be written in the mapping file of the given file (potentially replacing existing data). If NULL the mapping from the given file is used.

Author(s)

Jan Philipp Dietrich

See Also

setConfig

Examples

```
## Not run:
addMapping("regionmappingH12.csv")

## End(Not run)
```

---

cacheArgumentsHash *Tool: cacheArgumentsHash*

Description

Calculate hash from given function arguments for given call

Usage

```
cacheArgumentsHash(call, args = NULL)
```

Arguments

call	A function as a string or symbol. Passing a vector of functions is possible, but is only intended for corresponding read/correct/convert functions. If multiple functions in a vector define arguments with the same name but different default values only the default defined in the first function is considered.
args	A list of named arguments used to call the given function(s). If duplicates of arguments exists the first occurrence of the argument will be used.

Value

A hash representing the given arguments hash for the given call. NULL, if no argument deviates from the default argument settings.

Author(s)

Jan Philipp Dietrich

See Also

cachePut, cacheName, getNonDefaultArguments

Examples

```
madrat:::cacheArgumentsHash("madrat:::readTau", args = list(subtype = "historical"))
madrat:::cacheArgumentsHash("madrat:::readTau", args = list(subtype = "paper"))
calls <- c(madrat:::readTau, madrat:::convertTau)
madrat:::cacheArgumentsHash(calls, args = list(subtype = "historical"))
```

`cacheCleanup` *cacheCleanup*

Description

Delete files older than the specified number of days, based on file time metadata (per default atime = last access time).

Usage

```
cacheCleanup(
  daysThreshold,
  path = getConfig("cachefolder", verbose = FALSE),
  timeType = c("atime", "mtime", "ctime"),
  ask = TRUE,
  readlineFunction = readline
)
```

Arguments

`daysThreshold`	Files older than this many days are deleted/returned.
`path`	Path to where to look for old files.
`timeType`	Which file metadata time should be used. One of atime (last access time, default), mtime (last modify time), ctime (last metadata change).
`ask`	Whether to ask before deleting.
`readlineFunction`	Only needed for testing. A function to prompt the user for input.

Details

File time metadata is not available on all systems and the semantics are also system dependent, so please be careful and check that the correct files are deleted. This function will return a data.frame containing all files that would be deleted if the user answers 'n' to the question. If deleting files fails a warning is created.

Value

If the user answers 'n', a data.frame as returned by base::file.info, containing only files older than <daysThreshold> days.

cacheCopy *cacheCopy*

Description

Copy cache files which were used for a given preprocessing.

Usage

```
cacheCopy(file, target = NULL, filter = NULL)
```

Arguments

file	Path to a log file or content of a log as character vector.
target	Folder to which the files should be copied. If NULL no data is copied.
filter	Regular expression to filter the cache files shown in the log file.

Value

A vector of cache files which match the given log information and filter.

Author(s)

Jan Philipp Dietrich

cacheGet *Tool: cacheGet*

Description

Load fitting cache data (if available)

Usage

```
cacheGet(prefix, type, args = NULL, graph = NULL, ...)
```

Arguments

prefix	function prefix (e.g. "calc" or "read")
type	output type (e.g. "TauTotal")
args	a list of named arguments used to call the given function
graph	A madrat graph as returned by getMadratGraph. Will be created with getMadratGraph if not provided.
...	Additional arguments for getMadratGraph in case that no graph is provided (otherwise ignored)

Value

cached data, if cache is available, otherwise NULL

Author(s)

Jan Philipp Dietrich

See Also

cachePut, cacheName

Examples

```
madrat:::cacheGet("calc", "TauTotal", packages = "madrat")
```

---

cacheName *Tool: cacheName*

Description

Load fitting cache data (if available)

Usage

```
cacheName(
  prefix,
  type,
  args = NULL,
  graph = NULL,
  mode = "put",
  packages = getConfig("packages"),
  globalenv = getConfig("globalenv")
)
```

Arguments

prefix	function prefix (e.g. "calc" or "read")
type	output type (e.g. "TauTotal")
args	a list of named arguments used to call the given function
graph	A madrat graph as returned by getMadratGraph. Will be created with getMadratGraph if not provided.
mode	Context in which the function is used. Either "get" (loading) or "put" (writing). In case of "put" the potential file name is returned. When set to "get", a file name will only be returned if the file exists (otherwise NULL) and in combination which setConfig(forcecache=TRUE) even a cache file with deviating hash might get selected.

packages	A character vector with packages for which the available Sources/Calculations should be returned
globalenv	Boolean deciding whether sources/calculations in the global environment should be included or not

Value

Name of fitting cache file, if available, otherwise NULL

Note

`setConfig(forcecache=TRUE)` strongly affects the behavior of `cacheName`. In read model it will also return cache names with deviating hashes if no fitting cache file is found (in that case it will just return the newest one). In write mode the hash in the name will be left out since due to cache forcing it cannot be guaranteed that the cache file agrees with the state represented by the hash.

Author(s)

Jan Philipp Dietrich

See Also

`cachePut`, `cacheName`

Examples

```
madrat:::cacheName("calc", "TauTotal")
```

---

cachePut *Tool: cachePut*

Description

Save data to cache

Usage

```
cachePut(x, prefix, type, args = NULL, graph = NULL, ...)
```

Arguments

x	data that should be written to cache
prefix	function prefix (e.g. "calc" or "read")
type	output type (e.g. "TauTotal")
args	a list of named arguments used to call the given function
graph	A madrat graph as returned by `getMadratGraph` Will be created with `getMadratGraph` if not provided.
...	Additional arguments for `getMadratGraph` in case that no graph is provided (otherwise ignored)

Author(s)

Jan Philipp Dietrich

See Also

cachePut, cacheName

Examples

```
## Not run:
example <- 1
madrat:::cachePut(example, "calc", "Example", packages = "madrat")

## End(Not run)
```

---

calcOutput *calcOutput*

Description

Calculate a specific output for which a calculation function exists. The function is a wrapper for specific functions designed for the different possible output types.

Usage

```
calcOutput(
  type,
  aggregate = TRUE,
  file = NULL,
  years = NULL,
  round = NULL,
  signif = NULL,
  supplementary = FALSE,
  append = FALSE,
  warnNA = TRUE,
  na_warning = NULL,
  try = FALSE,
  regionmapping = NULL,
  writeArgs = NULL,
  ...
)
```

Arguments

type	output type, e.g. "TauTotal". A list of all available source types can be retrieved with function getCalculations.

aggregate	Boolean indicating whether output data aggregation should be performed or not, "GLO" (or "glo") for aggregation to one global region, "REG+GLO" (or "reg-glo") for a combination of regional and global data.
file	A file name. If given the output is written to that file in the outputfolder as specified in the config.
years	A vector of years that should be returned. If set to NULL all available years are returned.
round	Number of decimal places to round to. Ignored if NULL. See round() for details.
signif	Number of significant digits to round to. Ignored if NULL. See signif() for details.
supplementary	boolean deciding whether supplementary information such as weight should be returned or not. If set to TRUE a list of elements will be returned!
append	boolean deciding whether the output data should be appended in the existing file. Works only when a file name is given in the function call.
warnNA	boolean deciding whether NAs in the data set should create a warning or not
na_warning	deprecated, please use warnNA instead
try	if set to TRUE the calculation will only be tried and the script will continue even if the underlying calculation failed. If set to FALSE calculation will stop with an error in such a case. This setting will be overwritten by the global setting debug=TRUE, in which try will be always interpreted as TRUE.
regionmapping	alternative regionmapping to use for the given calculation. It will temporarily overwrite the global setting just for this calculation.
writeArgs	a list of additional, named arguments to be supplied to the corresponding write function
...	Additional settings directly forwarded to the corresponding calculation function

Value

magpie object with the requested output data either on country or on regional level depending on the choice of argument "aggregate" or a list of information if supplementary is set to TRUE.

Note

The underlying calc-functions are required to provide a list of information back to calcOutput. Following list entries should be provided:

- **x** - the data itself as magclass object
- **weight** - a weight for the spatial aggregation
- **unit** - unit of the provided data
- **description** - a short description of the data
- **note** (optional) - additional notes related to the data
- **class** (optional | default = "magpie") - Class of the returned object. If set to something other than "magpie" most functionality, such as aggregation or unit tests will not be available and is switched off!

- **isocountries** (optional | default = TRUE (mostly) or FALSE (if global)) - a boolean indicating whether data is in iso countries or not (the latter will deactivate several features such as aggregation)
- **mixed_aggregation** (optional | default = FALSE) - boolean which allows for mixed aggregation (weighted mean mixed with summations). If set to TRUE weight columns filled with NA will lead to summation, otherwise they will trigger an error.
- **min** (optional) - Minimum value which can appear in the data. If provided calcOutput will check whether there are any values below the given threshold and warn in this case
- **max** (optional) - Maximum value which can appear in the data. If provided calcOutput will check whether there are any values above the given threshold and warn in this case
- **structure.spatial** (optional) - regular expression describing the name structure of all names in the spatial dimension (e.g. `"^[A-Z]\{3\}$"`). Names will be checked against this regular expression and disagreements will be reported via a warning.
- **structure.temporal** (optional) - regular expression describing the name structure of all names in the temporal dimension (e.g. `"^y[0-9]\{4\}$"`). Names will be checked against this regular expression and disagreements will be reported via a warning.
- **structure.data** (optional) - regular expression describing the name structure of all names in the data dimension (e.g. `"^[a-z]*\\\\.[a-z]*$"`). Names will be checked against this regular expression and disagreements will be reported via a warning.
- **aggregationFunction** (optional | default = toolAggregate) - Function to be used to aggregate data from country to regions. The function must have the argument `x` for the data itself and `rel` for the relation mapping between countries and regions and must return the data as magpie object in the spatial resolution as defined in rel.
- **aggregationArguments** (optional) - List of additional, named arguments to be supplied to the aggregation function. In addition to the arguments set here, the function will be supplied with the arguments `x`, `rel` and if provided/deviating from the default also `weight` and `mixed_aggregation`.
- **putInPUC** (optional) boolean which decides whether this calculation should be added to a puc file which contains non-aggregated data and can be used to later on aggregate the data to resolutions of own choice. If not set `calcOutput` will try to determine automatically, whether a file is being required for the puc file or not, but in more complex cases (e.g. if calculations below top-level have to be run as well) this setting can be used to manually tweak the puc file list. CAUTION: Incorrect settings will cause corrupt puc files, so use this setting with extreme care and only if necessary.
- **cache** (optional) boolean which decides whether a cache file should be written (if caching is active) or not. Default setting is TRUE. This can be for instance useful, if the calculation itself is quick, but the corresponding file sizes are huge. Or if the caching for the given data type does not support storage in RDS format. CAUTION: Deactivating caching for a data set which should be part of a PUC file will corrupt the PUC file. Use with care.

Author(s)

Jan Philipp Dietrich

See Also

`setConfig`, `calcTauTotal`,

Examples

```
## Not run:

a <- calcOutput(type = "TauTotal")

## End(Not run)
```

`calcTauTotal` *Calculate total tau*

Description

This function prepares total tau values for use. As the source data already provides all required information this function purely removes not required data and moves xref values to the weighting object which is required for aggregation.

Usage

```
calcTauTotal(source = "paper")
```

Arguments

`source`	data source, either "paper" (default) or "historical".

Value

Total tau data and corresponding weights as a list of two MAgPIE objects

Author(s)

Jan Philipp Dietrich

See Also

`calcOutput`, `readTau`, `convertTau`

Examples

```
## Not run:
calcOutput("TauTotal")

## End(Not run)
```

compareData *compareData*

Description

Compares the content of two data archives and looks for similarities and differences

Usage

```
compareData(x, y, tolerance = 10^-5, yearLim = NULL)
```

Arguments

x	Either a tgz file or a folder containing data sets
y	Either a tgz file or a folder containing data sets
tolerance	tolerance level below which differences will get ignored
yearLim	year until when the comparison should be performed. Useful to check if data is identical until a certain year.

Author(s)

Jan Philipp Dietrich, Florian Humpenoeder

See Also

setConfig, calcTauTotal,

---

compareMadratOutputs *Compare a madrat function's output with and without your changes*

Description

With 'compareMadratOutputs' you can easily compare the output of a madrat function (read, calc, ...) with and without your changes. First, run 'compareMadratOutputs' without your changes, so a '.rds' file with the original output is saved. Then apply your changes and run 'compareMadratOutputs' again to compare the new output to the original output.

Usage

```
compareMadratOutputs(package, functionName, subtypes, overwriteOld = FALSE)
```

Arguments

package	[character(1)] The package where the given function is located. It will be attached via 'library'.
functionName	[character(1)] The name of the function from which you want to compare outputs. Must be a madrat function whose name starts with read, correct, convert, or calc.
subtypes	[character(n)] The subtypes you want to check. For calc functions this must be NULL.
overwriteOld	If TRUE: overwrite a "*-old-*.rds" previously created with compareMadratOutputs.

Details

If there are differences a '<functionName>-new.rds' containing the new output is saved for closer inspection. All files are created in the current working directory.

Value

Invisibly the result of 'waldo::compare' or 'all.equal' if a comparison was made, otherwise a named list of the outputs for each subtype.

Author(s)

Pascal Sauer

Examples

```
## Not run:
# save original output to readTau-old.rds
compareMadratOutputs("madrat", "readTau", c("paper", "historical"))

# now apply your changes to madrat:::readTau, reinstall madrat, restart the R session

# compare new output to original output from readTau-old.rds
compareMadratOutputs("madrat", "readTau", c("paper", "historical"))

## End(Not run)
```

---

convertTau *Convert Tau*

Description

Convert landuse intensity data (tau) to data on ISO country level.

Usage

```
convertTau(x)
```

Arguments

x	MAgPIE object containing tau values and corresponding weights xref at 0.5deg cellular level.

Value

Tau data and weights as MAgPIE object aggregated to country level

Author(s)

Jan Philipp Dietrich

---

downloadSource *downloadSource*

Description

Download a source. The function is a wrapper for specific functions designed for the different possible source types.

Usage

```
downloadSource(type, subtype = NULL, overwrite = FALSE, numberOfTries = 300)
```

Arguments

type	source type, e.g. "IEA". A list of all available source types can be retrieved with function `getSources("download")`.
subtype	For some sources there are subtypes of the source, for these source the subtype can be specified with this argument. If a source does not have subtypes, subtypes should not be set.
overwrite	Boolean deciding whether existing data should be overwritten or not.
numberOfTries	Integer determining how often readSource will check whether a running download is finished before exiting with an error. Between checks readSource will wait 30 seconds. Has no effect if the sources that should be read are not currently being downloaded.

Note

The underlying download-functions are required to provide a list of information back to `downloadSource`. Following list entries should be provided:

- **url** - full path to the file that should be downloaded
- **title** - title of the data source
- **author** - author(s) of the data set
- **license** - license of the data set. Put **unknown** if not specified.
- **description** - description of the data source
- **unit** - unit(s) of the data
- **doi** (optional) - a DOI URL to the data source
- **version** (optional) - version number of the data set
- **release_date** (optional) - release date of the data set
- **reference** (optional) - A reference for the data set (e.g. a paper, if the data was derived from it)

This user-provided data is enriched by automatically derived metadata:

- **call** - Information about the used madrat function call to download the data will check whether there are any values below the given threshold and warn in this case
- **accessibility** - A measure of quality for the accessibility of the data. Currently it distinguished between **iron** (manual access), **silver** (automatic access via URL) and **gold** (automatic access via DOI).

Besides the names above (user-provided and automatically derived) it is possible to add custom metadata entries by extending the return list with additional, named entries.

Author(s)

Jan Philipp Dietrich, David Klein, Pascal Sauer

See Also

`setConfig`, `readSource`

Examples

```
## Not run:
a <- downloadSource("Tau", subtype = "historical")

## End(Not run)
```

findBottlenecks *findBottlenecks*

Description

Analyzes a log from a retrieveData run, extracts runtime information for all called functions and identifies most critical bottlenecks.

Usage

```
findBottlenecks(file, unit = "min", cumulative = TRUE)
```

Arguments

file	path to a log file or content of a log as character vector
unit	unit for runtime information, either "s" (seconds), "min" (minutes) or "h" (hours)
cumulative	boolean deciding whether calls to the same function should be aggregated or not

Value

A data.frame sorted by net runtime showing for the different data processing functions their total runtime "time" (including the execution of all sub-functions) and net runtime "net" (excluding the runtime of sub-functions) and their share of total runtime.

Author(s)

Jan Philipp Dietrich

fingerprint *Tool: fingerprint*

Description

Function which creates a unique fingerprint for a madrat function based on the code of the function itself, other madrat functions which are called by this function and of all source folders involved in the process. The fingerprint can serve as an indication whether the workflow for the given function has been most likely changed, or not. If all involved source folders and the code of all involved functions remains the same, also the fingerprint will stay the same, otherwise it will change. Hence, it can be to figure out whether a cache file can be used for further calculations, or whether the calculation should be redone.

Usage

```
fingerprint(name, details = FALSE, graph = NULL, ...)
```

Arguments

name	Name of the function to be analyzed
details	Boolean indicating whether additional details in form of an attribute with underlying hash information should be added or not
graph	A madrat graph as returned by getMadratGraph. Will be created with getMadratGraph if not provided.
...	Additional arguments for getMadratGraph in case that no graph is provided (otherwise ignored)

Value

A fingerprint (hash) of all provided sources, or "fingerprintError"

Note

For a better performance only the first 300 bytes of each file and the corresponding file size is hashed. As the fingerprint function only takes madrat-based functions into account (e.g. read-functions or calc-functions), but does ignore all other functions there might be cases where calculations actually changed, but the fingerprint is still the same. In a similar fashion it is possible that the fingerprint changes even though the workflow stayed the same (as the dependencies are sometimes overestimated).

Author(s)

Jan Philipp Dietrich, Pascal Sauer

See Also

readSource

Examples

```
madrat:::fingerprint("toolGetMapping", package = "madrat")
```

---

fullEXAMPLE *fullExample*

Description

Example for class of fullX functions. Can be used as template for a new function or for testing the basic functionality

Usage

```
fullEXAMPLE(rev = 0, dev = "", extra = "Example argument")
```

Arguments

rev	data revision which should be used/produced. Will be converted to numeric_version when called via retrieveData.
dev	development suffix to distinguish development versions for the same data revision. This can be useful to distinguish parallel lines of development.
extra	additional argument which - when changed - does not require a re-computation of the portable unaggegrated collection (puc) file. setConfig (e.g. for setting the mainfolder if not already set properly).

Author(s)

Jan Philipp Dietrich

See Also

readSource,getCalculations,calcOutput,setConfig

Examples

```
## Not run:
retrieveData("example", rev = "2.1.2", dev = "test", regionmapping = "regionmappingH12.csv")

## End(Not run)
```

---

getCalculations *getCalculations*

Description

This function can be used to retrieve a list of currently available sources and outputs (based on the availability of corresponding conversion functions in the loaded data data processing packages.)

Usage

```
getCalculations(
  prefix = "calc",
  packages = getConfig("packages"),
  globalenv = getConfig("globalenv")
)
```

Arguments

prefix	Type of calculations, vector of types or search term (e.g. "read|calc"). Available options are "download" (source download), "read" (source read), "correct" (source corrections), "convert" (source conversion to ISO countries), "calc" (further calculations), and "full" (collections of calculations)
packages	A character vector with packages for which the available Sources/Calculations should be returned
globalenv	Boolean deciding whether sources/calculations in the global environment should be included or not

Value

A data frame containing all currently available outputs of all loaded data processing packages including its name, its function call and its package origin.

Author(s)

Jan Philipp Dietrich

See Also

readSource, setConfig

Examples

```
print(getCalculations())
print(getCalculations("read"))
```

---

getCode *getCode*

Description

Extract function code from madrat-style functions in specified packages

Usage

```
getCode(
  packages = installedMadratUniverse(),
  globalenv = getConfig("globalenv")
)
```

Arguments

packages	A character vector with packages for which the available Sources/Calculations should be returned
globalenv	Boolean deciding whether sources/calculations in the global environment should be included or not

Value

A named vector with condensed function code

Author(s)

Jan Philipp Dietrich

See Also

getMadratGraph

---

getConfig *getConfig*

Description

This function returns the madrat config which is currently loaded. If no configuration has been loaded so far the configuration will be initialized with default settings or system settings (if available).

Usage

```
getConfig(option = NULL, raw = FALSE, verbose = TRUE, print = FALSE)
```

Arguments

option	The option for which the setting should be returned. If set to NULL all options are returned.
raw	If set to FALSE some settings will be calculated, e.g. if the cache folder is set to FALSE the full path will be calculated using the main folder, or if the verbosity is not set the default verbosity will be returned. If raw is set to TRUE settings are returned as they are currently stored.
verbose	boolean deciding whether status information/updates should be shown or not
print	if TRUE and verbose is TRUE a configuration overview will also get printed

Value

A config list with all settings currently set for the madrat package

Note

getConfig is primarily designed to make the overall madrat configuration available to system tools of the madrat framework. There are only a few exceptions for which configuration settings are also readable from within a download-, read-, convert-, correct-, calc- or full-function. These exceptions are the setting "debug" (which can be used to add additional debug messages when active), the "tmpfolder" which can be used to temporarily store data and the setting "hash" (which can only be accessed from within a full function and can there be used to apply the identical hash algorithm for other calculations in which hashing is being used). Besides that "regionmapping" and "extramappings" can also be read from within calc- and full-functions but their use is at least for the calc-functions discouraged as it either might lead to incorrect caching behavior, or - if implemented correctly - lead to significant slow-downs of overall calculations. All other settings are currently still accessible but trigger a warning that this option will soon be removed. So, please make sure that your code runs without reading these options! As a background note: Read access to these settings will be restricted as they otherwise would allow access to code elements or data in a form which is violating the overall madrat logic and thereby can lead to erroneous results.

Author(s)

Jan Philipp Dietrich

See Also

setConfig, initializeConfig

---

getDependencies *getDependencies*

Description

Returns information about dependencies of a madrat-based calc- read- or full-function.

Usage

```
getDependencies(
  name,
  direction = "in",
  graph = NULL,
  type = NULL,
  self = FALSE,
  ...
)
```

Arguments

name	name of the function to be analyzed

direction	Character string, either “in”, “out”, "both", “full”, "din" or "dout". If “in” all sources feeding into the function are listed. If “out” consumer of the function are listed. If “both” the union of "in" and "out" is returned. If "full" the full network this function is connected to is shown, including indirect connections to functions which neither source nor consume the given function but serve as sources to other consumer functions. "din" and "dout" (short for "direct in" and "direct out") behave like "in" and "out" but only show direct calls in or from the function (ignoring the network of functions attached to it).
graph	A madrat graph as returned by getMadratGraph. Will be created with getMadratGraph if not provided.
type	type filter. Only dependencies of that type will be returned. Currently available types are "calc", "read" and "tool"
self	boolean defining whether the function itself, which is analyzed, should be included in the output, or not
...	Additional arguments for getMadratGraph in case that no graph is provided (otherwise ignored)

Author(s)

Jan Philipp Dietrich

See Also

getCalculations, getMadratGraph, getMadratInfo

---

getFlags *getFlags*

Description

Support function which extracts flags from code. Flags are string literals in a function body, for example ‘"!# @pucArguments extra"‘.

Usage

```
getFlags(code)
```

Arguments

code	A character vector with code from functions to be analyzed

Value

A list of found flag entries

Author(s)

Jan Philipp Dietrich

See Also

getCode

---

getISOlist *get official ISO list*

Description

Function which returns the ISO list which is used as default for the input data preparation. It contains the countries to which all source data has to be up- or downscaled to.

Usage

```
getISOlist(type = "all", threshold = 1)
```

Arguments

type	Determines what countries should be returned. "all" returns all countries, "important" returns all countries which are above the population threshold set in the configuration and "dispensable" returns all countries which are below the threshold.
threshold	Population threshold in million capita which determines whether the country is put into the "important" or "dispensable" class (default = 1 mio. people)

Value

vector of default ISO country codes.

Note

Please always use this function instead of directly referring to the data object as the format in this data list might change in the future!

Author(s)

Jan Philipp Dietrich

See Also

getSources, getCalculations

Examples

```
head(getISOlist())
head(getISOlist("dispensable"))
```

getLinkFunction *getLinkFunction*

Description

Returns a function that creates a symlink, hardlink, junction, or copy of files and directories, depending on OS capabilities (usually symlinks are not supported on Windows).

Usage

```
getLinkFunction()
```

Value

A function with arguments "from" and "to" which should behave like file.symlink on all platforms.

Author(s)

Pascal Sauer

getLocation *getLocation*

Description

Returns names of packages in which functions matching the description are found

Usage

```
getLocation(name, packages = installedMadratUniverse(), globalenv = TRUE)
```

Arguments

name	name of the function to be found. Can be either the full name (e.g. "calcTauTotal"), or just the type name (e.g. "TauTotal").
packages	A character vector with packages in which should be looked for the function
globalenv	Boolean deciding whether functions in the global environment should be included or not

Value

vector of packages in which a function matching the description could be found

Author(s)

Jan Philipp Dietrich

See Also

getCalculations, getDependencies

---

getMadratGraph *getMadratGraph*

Description

Function returns the madrat graph of all linkages of full, calc, and read functions of the given madrat based packages. Linkages to subfunctions of read functions (i.e. download, correct or convert functions) are not listed separately, but collectively referred to through the corresponding read function.

Usage

```
getMadratGraph(
  packages = installedMadratUniverse(),
  globalenv = getConfig("globalenv")
)
```

Arguments

packages	A character vector with packages for which the available Sources/Calculations should be returned
globalenv	Boolean deciding whether sources/calculations in the global environment should be included or not

Value

A data frame with 4 columns: from (source function), from_package (package the source function originates from), to (function which is using the source), to_package (package of the using function)

Author(s)

Jan Philipp Dietrich

See Also

getCalculations, getConfig

getMadratInfo *getMadratInfo*

Description

Collects and returns detailed information about the currently loaded network of madrat functions.

Usage

```
getMadratInfo(graph = NULL, cutoff = 5, extended = FALSE, ...)
```

Arguments

graph	A madrat graph as returned by getMadratGraph. Will be created with getMadratGraph if not provided.
cutoff	Integer introducing a cutoff of items to be returned for outputs which can become quite verbose.
extended	Will add additional outputs which has been removed from standard output due to limited usefulness.
...	Additional arguments for getMadratGraph in case that no graph is provided (otherwise ignored)

Author(s)

Jan Philipp Dietrich

See Also

getCalculations, getMadratGraph

getMadratMessage *getMadratMessage*

Description

Read a madrat message from the madrat environment. The madrat environment behaves similar like global options, except that 1) messages will also be stored in cache files and restored when a cache file is being loaded and 2) messages are always stored in lists with messages split by function calls where the message was triggered.

Usage

```
getMadratMessage(name = NULL, fname = NULL)
```

Arguments

name	The category in which the message should be stored
fname	function name. If specified only messages belonging to the functions history will be returned (this includes entries from the function itself, but also entries from functions which were called by this function).

Author(s)

Jan Philipp Dietrich

See Also

getMadratMessage

Examples

```
putMadratMessage("test", "This is a toast", fname = "readTau")
getMadratMessage("test", fname = "calcTauTotal")
```

---

getMainfolder *getMainfolder*

Description

Functions checks for a global setting of the mainfolder (either by setting the environment variable "MADRAT_MAINFOLDER" or by setting the R option with the same name). If none of these is available the user will be asked for a directory. If this is not provided a temporary folder will be used.

Usage

```
getMainfolder(verbose = TRUE, .testmode = FALSE)
```

Arguments

verbose	boolean deciding whether status information/updates should be shown or not
.testmode	boolean switch only relevant for internal testing (will simulate user inputs)

Author(s)

Jan Philipp Dietrich

See Also

initializeConfig, getConfig, setConfig

getNonDefaultArguments *getNonDefaultArguments*

Description

Given a function and an argument list, identify which arguments are different from their default.

Usage

```
getNonDefaultArguments(call, args = NULL)
```

Arguments

call	A function name as a string or symbol. Passing a vector of functions is possible, but is only intended for corresponding read/correct/convert functions. If multiple functions in a vector define arguments with the same name but different default values only the default defined in the first function is considered.
args	A list of named arguments used to call the given function(s). If duplicates of arguments exists the first occurrence of the argument will be used.

Value

A subset of args that is used by the function/s and is different from default values.

Author(s)

Jan Philipp Dietrich

See Also

cacheArgumentsHash, toolstartmessage

Examples

```
madrat:::getNonDefaultArguments("madrat:::readTau", args = list(subtype = "historical"))
madrat:::getNonDefaultArguments("madrat:::readTau", args = list(subtype = "paper"))
calls <- c(madrat:::readTau, madrat:::convertTau)
madrat:::getNonDefaultArguments(calls, args = list(subtype = "historical"))
```

`getSourceFolder` *getSourceFolder*

Description

Return the path to source data files for the given type and subtype. This applies redirections, see `redirectSource` for more details.

Usage

```
getSourceFolder(type, subtype)
```

Arguments

`type`	Dataset name, e.g. "Tau" for `readTau`
`subtype`	Subtype of the dataset, e.g. "paper" for `readTau`, NULL is allowed

Value

Path to source data files

Author(s)

Pascal Sauer

`getSources` *getSources*

Description

These functions can be used to retrieve a list of currently available sources and outputs (based on the availability of corresponding conversion functions in the loaded data data processing packages.)

Usage

```
getSources(
  name = NULL,
  type = NULL,
  packages = getConfig("packages"),
  globalenv = getConfig("globalenv")
)
```

Arguments

name	name of function for which sources should get returned. If not specified, all sources in the specified environment are returned
type	Type of source, either set to "read", "convert", "correct", "download" or NULL. If specified, a vector containing the sources with the corresponding function type are returned, otherwise a data.frame with all sources and their available function types is returned.
packages	A character vector with packages for which the available Sources/Calculations should be returned
globalenv	Boolean deciding whether sources/calculations in the global environment should be included or not

Value

A vector or data.frame containing all corresponding sources

Note

Please be aware that these functions only check the availability of corresponding functions of the package, not whether the functions will properly work.

Author(s)

Jan Philipp Dietrich

See Also

readSource, setConfig

Examples

```
print(getSources())
```

---

initializeConfig *initializeConfig*

Description

Checks whether configuration already has been set. If not, it will be initialized with default settings or (if available) system settings. All madrat folders (see setConfig for documentation which folders are available) will be set to the system environment variables MADRAT_SOURCEFOLDER, MADRAT_CACHEFOLDER, etc. if they exist, NA otherwise. NA means subfolders of the mainfolder are used.

Usage

```
initializeConfig(verbose = TRUE)
```

Arguments

verbose	boolean deciding whether status information/updates should be shown or not

Author(s)

Jan Philipp Dietrich

See Also

getMainfolder, getConfig, setConfig

---

installedMadratUniverse *installedMadratUniverse*

Description

Returns a name vector of installed packages which supposedly belong to the madrat universe. They are currently derived as the union of all loaded madrat packages and all packages with a name starting with "mr" or "ms" (as the usual indicator for madrat-packages and madrat-support-packages).

Usage

```
installedMadratUniverse()
```

Value

A name vector of installed packages which supposedly belong to the madrat universe

Author(s)

Jan Philipp Dietrich

See Also

setConfig

Examples

```
## Not run:
installedMadratUniverse()

## End(Not run)
```

`isWrapperActive` *isWrapperActive*

Description

Support functions which checks whether a given wrapper function is currently in-use or not or which locally activate or deactivate a wrapper (setting will be automatically resetted when a function finishes).

Usage

```
isWrapperActive(name)

setWrapperActive(name)

setWrapperInactive(name)
```

Arguments

`name`	name of the wrapper in question (e.g. "calcOutput")

Functions

- `setWrapperActive()`: set wrapper activity status to on
- `setWrapperInactive()`: set wrapper activity status to off

Author(s)

Jan Philipp Dietrich

`madapply` *madapply*

Description

This function is defunct and will be completely removed soon.

Usage

```
madapply(...)
```

Arguments

`...`	placeholder

Author(s)

Jan Philipp Dietrich

madlapply *madlapply*

Description

This function is defunct and will be completely removed soon.

Usage

```
madlapply(...)
```

Arguments

...	placeholder

Author(s)

Jan Philipp Dietrich

madratAttach *madratAttach / madratDetach*

Description

Attaches the madrat functions of a package to the currently active madrat universe or detaches it again from it.

Usage

```
madratAttach(package)

madratDetach(package)
```

Arguments

package	name of the package to be loaded. Alternative, the path to the package.

Functions

- `madratDetach()`: detach package from madrat universe

Author(s)

Jan Philipp Dietrich

See Also

getConfig, setConfig

Examples

```
## Not run:
madratAttach("madrat")

## End(Not run)
```

---

madTempDir *madTempDir*

Description

returns a temporary directory as a subfolder of the tempfolder set in `getConfig("tmpfolder")`.

Usage

```
madTempDir()
```

Value

path to the temp folder

Author(s)

Jan Philipp Dietrich

See Also

getConfig

Examples

```
## Not run:
madrat:::madTempDir()

## End(Not run)
```

metadataGFZ *metadataGFZ*

Description

Function to extract metadata information of a data set hosted at GFZ dataservices (https://dataservices.gfz-potsdam.de/portal/).

Usage

```
metadataGFZ(doi)
```

Arguments

doi	DOI of a data set hosted at GFZ dataservices

Value

a list with entries "license", "citation", "authors" and "year"

Author(s)

Jan Philipp Dietrich

See Also

toolstartmessage, vcat

Examples

```
## Not run:
metadataGFZ("10.5880/pik.2019.004")

## End(Not run)
```

prepFunctionName *prepFunctionName*

Description

Function to prepare a function call for a given type and prefix

Usage

```
prepFunctionName(type, prefix = "calc", ignore = NULL, error_on_missing = TRUE)
```

Arguments

type	name of calculation/source
prefix	Type of calculations. Available options are "download" (source download), "read" (source read), "correct" (source corrections), "convert" (source conversion to ISO countries), "calc" (further calculations), and "full" (collections of calculations)
ignore	vector of arguments which should be ignored (not be part of the function call)
error_on_missing	boolean deciding whether a missing type should throw an error or return NULL

Value

A function call as character to the specified function with corresponding package as attribute

Author(s)

Jan Philipp Dietrich

See Also

readSource, setConfig

Examples

```
print(madrat:::prepFunctionName("Tau","read"))
print(madrat:::prepFunctionName("TauTotal","calc"))
print(madrat:::prepFunctionName("EXAMPLE","full"))
```

---

pucAggregate *pucAggregate*

Description

Function which takes a puc-file ("portable unaggregated collection") as created via retrieveData and computes the corresponding aggregated collection with the provided arguments (e.g. the provided region mapping). The resulting tgz-file containing the collection will be put to the madrat outputfolder as defined in getConfig("outputfolder").

Usage

```
pucAggregate(
  puc,
  regionmapping = getConfig("regionmapping"),
  ...,
  renv = TRUE,
  strict = FALSE
)
```

Arguments

puc	path to a puc-file
regionmapping	region mapping to be used for aggregation.
...	(Optional) Settings that should be changed in addition. NOTE: which settings can be modified varies from puc to puc. Allowed settings are typically listed in the file name of the puc file after the revision number.
renv	Boolean which determines whether data should be aggregated from within a renv environment (recommended) or not. If activated, renv will check which packages in which versions were used to create the puc file, download, install and load these packages and run the aggregation with them. Otherwise, the packages in the currently used environment are being used.
strict	Boolean or NULL which allows to trigger a strict mode. During strict mode warnings will be taken more seriously and will cause 1. to have the number of warnings as prefix of the created tgz file and 2. will prevent retrieveData from creating a puc file. If set to NULL the setting will be read from the puc file.

Author(s)

Jan Philipp Dietrich

See Also

retrieveData,localConfig

Examples

```
## Not run:
pucAggregate("rev1_example.puc", regionmapping = "regionmappingH12.csv")

## End(Not run)
```

---

putMadratMessage *putMadratMessage*

Description

Store a madrat message in the madrat environment. The madrat environment behaves similar like global options, except that 1) messages will also be stored in cache files and restored when a cache file is being loaded and 2) messages are always stored in lists with messages split by function calls where the message was triggered.

Usage

```
putMadratMessage(name, value, fname = -1, add = FALSE)
```

Arguments

name	The category in which the message should be stored
value	The message that should be recorded as character. Alternatively, if name is not set, it is also possible to provide a complete list of the structure value[[name]][[fname]] where name and fname correspond to the category name and function name entries (e.g. value = list(test = list(readTau = "This is a toast"))).
fname	function name the entry belongs to or the frame number from which the function name should be derived from (e.g. -1 to recieve function name from parent function).
add	boolean deciding whether the value should be added to a existing value (TRUE) or overwrite it (FALSE)

Author(s)

Jan Philipp Dietrich

See Also

putMadratMessage

Examples

```
putMadratMessage("test", "This is a toast", fname = "readTau")
getMadratMessage("test", fname = "calcTauTotal")
```

---

readSource *readSource*

Description

Read in a source file and convert it to a MAgPIE object. The function is a wrapper for specific functions designed for the different possible source types.

Usage

```
readSource(
  type,
  subtype = NULL,
  subset = NULL,
  convert = TRUE,
  supplementary = FALSE
)
```

Arguments

type	A character string referring to the source type, e.g. "IEA" which would internally call a function called ‘readIEA‘ (the "wrapped function"). A list of available source types can be retrieved with function getSources.
subtype	A character string. For some sources there are subtypes of the source, for these sources the subtype can be specified with this argument. If a source does not have subtypes, subtypes should not be set.
subset	A character string. Similar to subtype a source can also have subsets. A subsets can be used to only read part of the data. This can in particular make sense for huge data sets where reading in the whole data set might be impractical or even infeasible.
convert	Boolean indicating whether input data conversion to ISO countries should be done or not. In addition it can be set to "onlycorrect" for sources with a separate correctXXX-function.
supplementary	Boolean deciding whether a list including the actual data and metadata, or just the actual data is returned.

Value

The read-in data, usually a magpie object. If supplementary is TRUE a list including the data and metadata is returned instead. The temporal and data dimensionality should match the source data. The spatial dimension should either match the source data or, if the convert argument is set to TRUE, should be on ISO code country level.

Note

If a magpie object is returned magclass::clean_magpie is run and if convert = TRUE ISO code country level is checked.

Author(s)

Jan Philipp Dietrich, Anastasis Giannousakis, Lavinia Baumstark, Pascal Sauer

See Also

setConfig, downloadSource, readTau #’ @note The underlying read-functions can return a magpie object or a list of information (preferred) back to readSource. In list format the object should have the following structure:

- **x** - the data itself as magclass object
- **unit** (optional) - unit of the provided data
- **description** (otional) - a short description of the data
- **note** (optional) - additional notes related to the data
- **class** (optional | default = "magpie") - Class of the returned object. If set to something other than "magpie" most functionality will not be available and is switched off!

- **cache** (optional) boolean which decides whether a cache file should be written (if caching is active) or not. Default setting is TRUE. This can be for instance useful, if the calculation itself is quick, but the corresponding file sizes are huge. Or if the caching for the given data type does not support storage in RDS format. CAUTION: Deactivating caching for a data set which should be part of a PUC file will corrupt the PUC file. Use with care.

Examples

```
## Not run:
a <- readSource("Tau", "paper")

## End(Not run)
```

`readTau` *Read Tau*

Description

Read-in landuse intensity data (tau) following the methodology published in Dietrich J.P., Schmitz C., Mueller C., Fader M., Lotze-Campen H., Popp A., Measuring agricultural land-use intensity - A global analysis using a model-assisted approach, Ecological Modelling, Volume 232, 10 May 2012, Pages 109-118, ISSN 0304-3800, 10.1016/j.ecolmodel.2012.03.002.

Usage

```
readTau(subtype = "paper")
```

Arguments

`subtype`	Type of Tau data that should be read. Available types are: • `paper`: numbers as they are reported in the paper (cellular, crop-specific) • `historical`: historic tau values on iso country level for total tau factor. This numbers were calculated by taking FAO yields and norming it to the 1995 tau values of the paper (faoyields*tau95/mean(faoyields[1995:2005]))

Value

Tau data and weights as MAgPIE object in original resolution

Author(s)

Jan Philipp Dietrich

See Also

`readSource`

Examples

```
## Not run:
a <- readSource("Tau")

## End(Not run)
```

`redirect` *redirect*

Description

Redirect a given dataset type to a different source folder. The redirection is local, so it will be reset when the current function call returns. See example for more details.

Usage

```
redirect(type, target, linkOthers = TRUE, local = TRUE)
```

Arguments

`type`	Dataset name, e.g. "Tau" for `readTau`
`target`	Either path to the new source folder that should be used instead of the default, or NULL to remove the redirection, or a vector of paths to files which are then symlinked into a temporary folder that is then used as target folder; if the vector is named the names are used as relative paths in the temporary folder, e.g. target = c(‘a/b/c.txt‘ = "~/d/e/f.txt") would create a temporary folder with subfolders a/b and there symlink c.txt to ~/d/e/f.txt.
`linkOthers`	If target is a list of files, whether to symlink all other files in the original source folder to the temporary folder.
`local`	The scope of the redirection, passed on to setConfig. Defaults to the current function. Set to an environment for more control or to FALSE for a permanent/global redirection.

Value

Invisibly, the source folder that is now used for the given type

Author(s)

Pascal Sauer

Examples

```
## Not run:
f <- function() {
  redirect("Tau", target = "~/TauExperiment")
  # the following call will change directory
  # into ~/TauExperiment instead of <getConfig("sourcefolder")>/Tau
  readSource("Tau")
}
f()
# Tau is only redirected in the local environment of f,
# so it will use the usual source folder here
readSource("Tau")

## End(Not run)
```

`redirectSource` *redirectSource*

Description

redirectSource will call a source specific redirect function if it exists (called e.g. redirectTau), in which case the arguments are passed on to that function. If such a function is not available `redirect` is called.

Usage

```
redirectSource(type, target, ..., linkOthers = TRUE, local = TRUE)
```

Arguments

`type`	The source dataset type. Passed on to the specific redirect function or `redirect`.
`target`	The target folder or files. Passed on to the specific redirect function or `redirect`.
`...`	Additional arguments, passed on to the specific redirect function.
`linkOthers`	Passed on to the specific redirect function or `redirect`.
`local`	Passed on to the specific redirect function or `redirect`.

Value

The result of the specific redirect function or `redirect`.

Author(s)

Pascal Sauer

redirectTau *redirectTau*

Description

redirectTau will be called by redirectSource when type = "Tau". Redirects the Tau source folder to the target folder.

Usage

```
redirectTau(target, ...)
```

Arguments

target	The target folder or files.
...	Passed on to redirect.

Author(s)

Pascal Sauer

Examples

```
## Not run:
redirectSource("Tau", "a/different/tau-source-folder")
a <- readSource("Tau", "paper")

## End(Not run)
```

regionscode *Tool: regionscode*

Description

Given a regionmapping (mapping between ISO countries and regions) the function calculates a regionscode which is basically the md5sum of a reduced form of the mapping. The regionscode is unique for each regionmapping and can be used to clearly identify a given regionmapping. In addition several checks are performed to make sure that the given input is a proper regionmapping

Usage

```
regionscode(mapping = NULL, label = FALSE, strict = TRUE)
```

Arguments

mapping	Either a path to a mapping or an already read-in mapping as data.frame. If set to NULL (default) the regionscode of the region mapping set in the madrat config will be returned.
label	logical deciding whether the corresponding label of a regionscode should be returned instead of the regionscode.
strict	If set to TRUE region mappings with mapping to ISO countries with exactly 2 columns or more than 2 colums (if the first colum contains irrelevant information which will be deleted automatically) will be accepted. In this case data will be transformed and even cases with different ordering will yield the same regionscode. If set to FALSE all these checks will be ignored and the regionscode will be just computed on the object as it is. Please be aware the regionscode will differ with strict mode on or off!

Value

A md5-based regionscode which describes the given mapping or, if label=TRUE and a corresponding label is available, the label belonging to the regionscode

Author(s)

Jan Philipp Dietrich

See Also

toolCodeLabels, fingerprint, digest

Examples

```
file <- system.file("extdata", "regionmappingH12.csv", package = "madrat")
regionscode(file)
```

---

resetMadratMessages *resetMadratMessages*

Description

Delete stored madrat messages from the madrat environment. The madrat environment behaves similar like global options, except that 1) messages will also be stored in cache files and restored when a cache file is being loaded and 2) messages are always stored in lists with messages split by function calls where the message was triggered.

Usage

```
resetMadratMessages(name = NULL, fname = NULL)
```

Arguments

name	The category for which the messages should be reset (if not set messages in all categories will be reset)
fname	function name for which the entries should be reset (if not specified messages for all function names will be reset)

Author(s)

Jan Philipp Dietrich

See Also

putMadratMessage, getMadratMessage

Examples

```
putMadratMessage("test", "This is a toast", fname = "readTau")
getMadratMessage("test", fname = "calcTauTotal")
resetMadratMessages("test")
```

---

retrieveData *retrieveData*

Description

Function to retrieve a predefined collection of calculations for a specific regionmapping.

Usage

```
retrieveData(
  model,
  rev = 0,
  dev = "",
  cachetype = "def",
  puc = identical(dev, ""),
  strict = FALSE,
  renv = TRUE,
  ...
)
```

Arguments

model	The names of the model for which the data should be provided (e.g. "magpie").
rev	data revision which should be used/produced. Will be converted to numeric_version.
dev	development suffix to distinguish development versions for the same data revision. This can be useful to distinguish parallel lines of development.

cachetype	defines what cache should be used. "rev" points to a cache shared by all calculations for the given revision and sets forcecache to TRUE, "def" points to the cache as defined in the current settings and does not change forcecache setting.
puc	Boolean deciding whether a fitting puc file (if existing) should be read in and if a puc file (if not already existing) should be created.
strict	Boolean which allows to trigger a strict mode. During strict mode warnings will be taken more seriously and will cause 1. to have the number of warnings as prefix of the created tgz file and 2. will prevent retrieveData from creating a puc file.
renv	Boolean which determines whether calculations should run within a renv environment (recommended) or not (currently only applied in pucAggregate). If activated, renv will check which packages in which versions were used to create the puc file, download, install and load these packages and run the aggregation with them. Otherwise, the packages in the currently used environment are being used.
...	(Optional) Settings that should be changed using setConfig (e.g. regionmapping). or arguments which should be forwarded to the corresponding fullXYZ function (Please make sure that argument names in full functions do not match settings in setConfig!)

Value

Invisibly, the path to the newly created tgz archive.

Note

The underlying full-functions can optionally provide a list of information back to retrieveData. Following list entries are currently supported:

- **tag** (optional) - additional name tag which will be included in the file name of the aggregated collection (resulting tgz-file). This can be useful to highlight information in the file name which otherwise would not be visible.
- **pucTag** (optional) - identical purpose as **tag** but for the resulting unaggregated collections (puc-files).

Author(s)

Jan Philipp Dietrich, Lavinia Baumstark

See Also

calcOutput,setConfig

Examples

```
## Not run:
retrieveData("example", rev = "2.1.1", dev = "test", regionmapping = "regionmappingH12.csv")

## End(Not run)
```

`robustOrder` *robustOrder, robustSort*

Description

robustOrder: A wrapper around base::order that always uses the locale independent method = "radix". If the argument x is a character vector it is converted to utf8 first. robustSort: A convenience function using order to sort a vector using radix sort. The resulting vector will have the same encoding as the input although internally character vectors are converted to utf8 before ordering.

Usage

```
robustOrder(..., na.last = TRUE, decreasing = FALSE, method = "radix")
```

Arguments

`...`	One or more vectors of the same length
`na.last`	If TRUE missing values are put last, if FALSE they are put first, if NA they are removed
`decreasing`	If TRUE decreasing/descending order, if FALSE increasing/ascending order. For the "radix" method, this can be a vector of length equal to the number of arguments in ... . For the other methods, it must be length one.
`method`	Default is "radix", which is locale independent. The alternatives "auto" and "shell" should not be used in madrat because they are locale dependent.

Author(s)

Pascal Sauer

See Also

`order`

`setConfig` *setConfig*

Description

This function manipulates the current madrat configuration. In general, NULL means that the argument remains as it is whereas all other inputs will overwrite the current setting. For values which can be reset to NULL (currently only "extramappings") you can achieve a reset by setting the value to "".

Usage

```
setConfig(
  regionmapping = NULL,
  extramappings = NULL,
  packages = NULL,
  globalenv = NULL,
  enablecache = NULL,
  verbosity = NULL,
  mainfolder = NULL,
  sourcefolder = NULL,
  cachefolder = NULL,
  mappingfolder = NULL,
  outputfolder = NULL,
  pucfolder = NULL,
  tmpfolder = NULL,
  nolabels = NULL,
  forcecache = NULL,
  ignorecache = NULL,
  cachecompression = NULL,
  hash = NULL,
  diagnostics = NULL,
  debug = NULL,
  maxLengthLogMessage = NULL,
  redirections = NULL,
  .cfgchecks = TRUE,
  .verbose = TRUE,
  .local = FALSE
)

localConfig(...)
```

Arguments

regionmapping	The name of the csv file containing the region mapping that should be used for aggregation (e.g. "regionmappingREMIND.csv").
extramappings	Names of additional mappings supplementing the given region mapping. This allows for additional aggregation levels such as subnational aggregation.
packages	A character vector with packages in which corresponding read and calc functions should be searched for
globalenv	Boolean deciding whether sources/calculations in the global environment should be included or not
enablecache	Is deprecated and will be ignored. Please use ignorecache instead.
verbosity	an integer value describing the verbosity of the functions (2 = full information, 1 = only warnings and execution information, 0 = only warnings, -1 = no information)
mainfolder	The mainfolder where all data can be found and should be written to.

`sourcefolder` The folder in which all source data is stored (in sub-folders with the name of the source as folder name). In the default case this argument is set to NA meaning that the default folder should be used which is <mainfolder>/sources

`cachefolder` The folder in which all cache files should be written to. In the default case this argument is set to NA meaning that the default folder should be used which is <mainfolder>/cache

`mappingfolder` A folder containing all kinds of mappings (spatial, temporal or sectoral). In the default case this argument is set to NA meaning that the default folder should be used which is <mainfolder>/mappings

`outputfolder` The folder all outputs should be written to. In the default case this argument is set to NA meaning that the default folder should be used which is <mainfolder>/output

`pucfolder` The path where portable unaggregated collection (puc) files are located. NA by default, which means <mainfolder>/puc

`tmpfolder` Path to a temp folder for temporary storage of files. By default set to <mainfolder>/tmp

`nolabels` vector of retrieve models (e.g. "EXAMPLE" in case of "fullEXAMPLE") which should NOT apply a replacement of known hashes with given code labels

`forcecache` Argument that allows to force madrat to read data from cache if the corresponding cache files exist. It is either a boolean to fully activate or deactivate the forcing or a vector of files (e.g. readTau, calcTauTotal) or type (e.g. Tau, TauTotal) that should be read from cache in any case.

`ignorecache` Argument that allows madrat to ignore the forcecache argument for the given vector of files (e.g. readTau, calcTauTotal) or types (e.g. Tau, TauTotal) called by calcOutput or readSource. The top level function must always be part of this list.

`cachecompression` logical or character string specifying whether cache files use compression. TRUE corresponds to gzip compression, and character strings "gzip", "bzip2" or "xz" specify the type of compression.

`hash` specifies the used hashing algorithm. Default is "xxhash32" and all algorithms supported by `digest` can be used.

`diagnostics` Either FALSE (default) to avoid the creation of additional log files or a file name for additional diagnostics information (without file ending).

`debug` Boolean which activates a debug mode. In debug mode all calculations will be executed with try=TRUE so that calculations do not stop even if the previous calculation failed. This can be helpful to get a full picture of errors rather than only seeing the first one. In addition debug=TRUE will add the suffix "debug" to the files created to avoid there use in productive runs. Furthermore, with debug=TRUE calculations will be rerun even if a corresponding tgz file already exists.

`maxLengthLogMessage` in log messages evaluated arguments are printed if the resulting message is shorter than this value, otherwise arguments are shown as passed, potentially with unevaluated variable names

redirections	A list of source folder redirections, intended to be set by redirectSource. See that function's documentation for more details.
.cfgchecks	boolean deciding whether the given inputs to setConfig should be checked for consistency or just be accepted (latter is only necessary in very rare cases and should not be used in regular cases)
.verbose	boolean deciding whether status information/updates should be shown or not
.local	boolean deciding whether options are only changed until the end of the current function execution OR environment for which the options should get changed.
...	Arguments forwarded to setConfig

Functions

- localConfig(): A wrapper for setConfig(..., .local = TRUE)

Note

setConfig must only be used before the data processing is started and changes in the configuration from within a download-, read-, correct-, convert-, calc-, or full-function are not allowed! Only allowed configuration update is to add another extramapping via addMapping. Currently the use of setConfig within any of these functions will trigger a warning, which is planned to be converted into an error message in one of the next package updates!

Author(s)

Jan Philipp Dietrich

See Also

getConfig, getISOlist

Examples

```
## Not run:
setConfig(forcecache = c("readSSPall", "convertSSPall"))

## End(Not run)
```

---

toolAggregate *toolAggregate*

Description

(Dis-)aggregates a magclass object from one resolution to another based on a relation matrix or mapping

Usage

```
toolAggregate(
  x,
  rel,
  weight = NULL,
  from = NULL,
  to = NULL,
  dim = 1,
  wdim = NULL,
  partrel = FALSE,
  negative_weight = "warn",
  mixed_aggregation = FALSE,
  verbosity = 1,
  zeroWeight = "warn"
)
```

Arguments

x	magclass object that should be (dis-)aggregated
rel	relation matrix, mapping or file containing a mapping in a format supported by toolGetMapping (currently csv, rds or rda). A mapping object consists of any number of columns, where one column contains all the elements in x. These elements are mapped to the corresponding values in another column, as described below (see parameter 'from'). It is possible to not set rel as long as to is set and dim is chosen appropriately. In that case the relation mapping is extracted from the dimnames of the corresponding dimension, e.g. if your data contains a spatial subdimension "country" you can aggregate to countries via toolAggregate(x, to = "country", dim = 1).
weight	magclass object containing weights which should be considered for a weighted aggregation. The provided weight should only contain positive values, but does not need to be normalized (any positive number>=0 is allowed). Please see the "details" section below for more information.
from	Name of source column to be used in rel if it is a mapping (if not set the first column matching the data will be used).
to	Name of the target column to be used in rel if it is a mapping (if not set the column following column from will be used If column from is the last column, the column before from is used). If data should be aggregated based on more than one column these columns can be specified via "+", e.g. "region+global" if the data should be aggregated to column regional as well as column global. If rel is missing to refers to the aggregation target dimension name.
dim	Specifying the dimension of the magclass object that should be (dis-)aggregated. Either specified as an integer (1=spatial,2=temporal,3=data) or if you want to specify a sub dimension specified by name of that dimension or position within the given dimension (e.g. 3.2 means the 2nd data dimension, 3.8 means the 8th data dimension).

wdim — Specifying the according weight dimension as chosen with dim for the aggregation object. If set to NULL the function will try to automatically detect the dimension.

partrel — If set to TRUE allows that the relation matrix does contain less entries than x and vice versa. These values without relation are lost in the output.

negative_weight — Describes how a negative weight should be treated. "allow" means that it just should be accepted (dangerous), "warn" returns a warning and "stop" will throw an error in case of negative values

mixed_aggregation — boolean which allows for mixed aggregation (weighted mean mixed with summations). If set to TRUE weight columns filled with NA will lead to summation.

verbosity — Verbosity level of messages coming from the function: -1 = error, 0 = warning, 1 = note, 2 = additional information, >2 = no message

zeroWeight — Describes how a weight sum of 0 for a category/aggregation target should be treated. "allow" accepts it and returns 0 (dangerous), "setNA" returns NA, "warn" throws a warning, "stop" throws an error.

Details

Basically toolAggregate is doing nothing more than a normal matrix multiplication which is taking into account the 3 dimensional structure of MAgPIE objects. So, you can provide any kind of relation matrix you would like. However, for easier usability it is also possible to provide weights for a weighted (dis-)aggregation as a MAgPIE object. In this case rel must be a 1-0-matrix or a mapping between both resolutions. The weight needs to be provided in the higher spatial aggregation, meaning for aggregation the spatial resolution of your input data and in the case of disaggregation the spatial resolution of your output data. The temporal and data dimension must be either identical to the resolution of the data set that should be (dis-)aggregated or 1. If the temporal and/or data dimension is 1 this means that the same transformation matrix is applied for all years and/or all data columns. In the case that a column should be just summed up instead of being calculated as a weighted average you either do not provide any weight (then all columns are just summed up) or your set this specific weighting column to NA and mixed_aggregation to TRUE.

Value

the aggregated data in magclass format

Author(s)

Jan Philipp Dietrich, Ulrich Kreidenweis

See Also

calcOutput

Examples

```
# create example mapping
p <- magclass::maxample("pop")
mapping <- data.frame(from = magclass::getItems(p, dim = 1.1),
                      region = rep(c("REG1", "REG2"), 5),
                      global = "GLO")
print(mapping)

# run aggregation
toolAggregate(p, mapping)
# weighted aggregation
toolAggregate(p, mapping, weight = p)
# combined aggregation across two columns
toolAggregate(p, mapping, to = "region+global")
```

---

toolCodeLabels *Tool: CodeLabels*

Description

This function replaces a hash code (e.g. regioncode) or another cryptic code with a human readable code via a given dictionary. This can be useful to make outputs better readable in cases where hash codes are already known to the user. If no entry exists in the dictionary the hash code is returned again.

Usage

```
toolCodeLabels(get = NULL, add = NULL)
```

Arguments

get	A vector of hash codes which should be replaced
add	Additional entries that should be added to the dictionary. Need to be provided in the form of a named vector with the structure c(<label>=<hash>), e.g. c(h12="62eff8f7")

Value

A vector with either labels (if available) or hash codes (if no label was available).

Author(s)

Jan Philipp Dietrich

See Also

regionscode

Examples

```
toolCodeLabels("62eff8f7")
```

`toolConditionalReplace` *toolConditionalReplace*

Description

Sets values (NA, negative, ..) to value `replaceby`

Usage

```
toolConditionalReplace(x, conditions, replaceby = 0)
```

Arguments

`x`	magpie object
`conditions`	vector of conditions for values, that should be removed e.g. "is.na()", "< 0" (order matters)
`replaceby`	value which should be used instead (can be a vector of same length as conditions as well)

Value

return changed input data

Author(s)

Kristine Karstens

`toolConvertMapping` *Tool: ConvertMapping*

Description

Function which converts mapping files between formats

Usage

```
toolConvertMapping(name, format = "rds", type = NULL, where = "mappingfolder")
```

Arguments

name	File name of the mapping file. Supported file types are currently csv (, or ; separated), rds and rda (which needs to have the data stored with the object name "data"!).
format	format it should be converted to. Available is "csv", "rds" or "rda".
type	Mapping type (e.g. "regional", "cell", or "sectoral"). Can be set to NULL if file is not stored in a type specific subfolder
where	location to look for the mapping, either "mappingfolder" or the name of a package which contains the mapping

Author(s)

Jan Philipp Dietrich

See Also

calcOutput, toolConvertMapping

---

toolCountry2isocode *toolCountry2isocode*

Description

Function used to convert country names from the long name to the ISO 3166-1 alpha 3 country code

Usage

```
toolCountry2isocode(
  country,
  warn = TRUE,
  ignoreCountries = NULL,
  type = NULL,
  mapping = NULL
)
```

Arguments

country	A vector of country names
warn	whether warnings should be printed now or in the end of the whole process as notes
ignoreCountries	A vector of country names/codes that exist in the data and that should be removed but without creating a warning (they will be removed in any case). You should use that argument if you are certain that the given entries should be actually removed from the data.
type	deprecated and will be removed soon!
mapping	additional mappings as a names vector

Value

the ISO 3166-1 alpha 3 country code

Author(s)

Jan Philipp Dietrich, Anastasis Giannousakis

See Also

readSource,getSources

Examples

```
toolCountry2isocode("Germany")
toolCountry2isocode(c("Germany","Fantasyland"),mapping=c("Fantasyland"="BLA"))
```

---

toolCountryFill *Tool: CountryFill*

Description

This function expects a MAgPIE object with ISO country codes in the spatial dimension. These ISO codes are compared with the official ISO code country list (stored as supplementary data in the madrat package). If there is an ISO code in the data but not in the official list this entry is removed, if an entry of the official list is missing in the data this entry is added and set to the value of the argument fill.

Usage

```
toolCountryFill(
  x,
  fill = NA,
  no_remove_warning = NULL,
  overwrite = FALSE,
  verbosity = 1,
  countrylist = NULL,
  ...
)
```

Arguments

x	MAgPIE object with ISO country codes in the spatial dimension
fill	Number which should be used for filling the gaps of missing countries.
no_remove_warning	A vector of non-ISO country codes that exist in the data and that should be removed by CountryFill but without creating a warning (they will be removed in any case). You should use that argument if you are certain that the given entries should be actually removed from the data.

overwrite	logical deciding whether existing data should be overwritten, if there is a specific mapping provided for that country, or not
verbosity	verbosity for information about filling important countries. 0 = warning will show up (recommended if filling of important countries is not expected), 1 = note will show up in reduced log file (default), 2 = info will show up in extended log file (recommended if filling of important countries is not critical and desired).
countrylist	character vector of official country names (if other than ISO)
...	Mappings between countries for which the data is missing and countries from which the data should be used instead for these countries (e.g. "HKG"="CHN" if Hong Kong should receive the value of China). This replacement usually only makes sense for intensive values. Can be also provided as a argument called "map" which contains a named vector of these mappings.

Value

A MAgPIE object with spatial entries for each country of the official ISO code country list.

Author(s)

Jan Philipp Dietrich

Examples

```
library(magclass)
x <- new.magpie("DEU", 1994, "bla", 0)
y <- toolCountryFill(x, 99)
```

---

toolendmessage *Tool: End message*

Description

This function writes a process end message and performs some diagnostics. It is always called after a corresponding call to toolstartmessage.

Usage

```
toolendmessage(startdata, level = NULL)
```

Arguments

startdata	a list containing diagnostic information provided by toolstartmessage
level	This argument allows to establish a hierarchy of print statements. The hierarchy is preserved for the next vcat executions. Currently this setting can have 4 states: NULL (nothing will be changed), 0 (reset hierarchies), "+" (increase hierarchy level by 1) and "-" (decrease hierarchy level by 1).

Author(s)

Jan Philipp Dietrich

See Also

toolstartmessage, vcat

`toolFillWithRegionAvg` *Tool: FillWithRegionAvg*

Description

This function fills missing values for countries with the (weighted) average of the respective region. The average is computed separately for every timestep. Currently only inputs with one data dimension are allowed as inputs. (If the filling should be performed over multiple data dimensions, call this function multiple times and bind the results together with magclass::mbind.)

Usage

```
toolFillWithRegionAvg(
  x,
  valueToReplace = NA,
  weight = NULL,
  callToolCountryFill = FALSE,
  regionmapping = NULL,
  verbose = TRUE,
  warningThreshold = 0.5,
  noteThreshold = 1
)
```

Arguments

`x`	MAgPIE object with country codes in the first and time steps in the second dimension.
`valueToReplace`	value that denotes missing data. Defaults to NA.
`weight`	MAgPIE object with weights for the weighted average. Must contain at least all the countries and years present in x. If no weights are specified, an unweighted average is performed.
`callToolCountryFill`	Boolean variable indicating whether the list of countries should first be filled to the official ISO code country list. Subsequently the newly added and previously missing values are filled with the region average.
`regionmapping`	Data frame containing the mapping between countries and regions. Expects column names CountryCode and RegionCode. Uses the currently set mapping if no mapping is specified.

verbose	Boolean variable indicating if the function should print out what it is doing. Can generate a lot of output for a large object.
warningThreshold	If more than this fraction of the countries in a given region and timestep have a missing value, throw a warning.
noteThreshold	If more than this fraction of the countries in a given region and timestep have a missing value, a note will be written.

Details

toolFillWithRegionAvg can be used in conjunction with toolCountryFill() to first fill up the list of countries to the official ISO code country list, and then fill values with the regional average (see callToolCountryFill Option).

Value

A MAgPIE object with the missing values filled.

Author(s)

Bjoern Soergel, Lavinia Baumstark, Jan Philipp Dietrich

Examples

```
x <- magclass::new.magpie(cells_and_regions = c("A", "B", "C", "D"), years = c(2000, 2005),
  fill = c(1, NA, 3, 4, 5, 6, NA, 8))
rel <- data.frame(CountryCode = c("A", "B", "C", "D"), RegionCode = c("R1", "R1", "R1", "R2"))
xfilled <- toolFillWithRegionAvg(x, regionmapping = rel)
```

---

toolFillYears *toolFillYears*

Description

Inter- and extrapolates a historical dataset for a given time period.

Usage

```
toolFillYears(x, years)
```

Arguments

x	MAgPIE object to be continued.
years	vector of years as digits or in mag year format

Value

MAgPIE object with completed time dimensionality.

Author(s)

Kristine Karstens

`toolGetMapping` *Tool: GetMapping*

Description

Function which retrieves a mapping file

Usage

```
toolGetMapping(
  name,
  type = NULL,
  where = NULL,
  error.missing = TRUE,
  returnPathOnly = FALSE,
  activecalc = NULL
)
```

Arguments

`name`	File name of the mapping file. Supported file types are currently csv (, or ; separated), rds and rda (which needs to have the data stored with the object name "data"!). Use `toolConvertMapping` to convert between both formats
`type`	Mapping type (e.g. "regional", "cell", or "sectoral"). Can be set to NULL if file is not stored in a type specific subfolder
`where`	location to look for the mapping, either "mappingfolder", "local" (if the path is relative to your current directory) or the name of a package which contains the mapping. If set to NULL it will first try "local", then "mappingfolder" and afterwards scan all packages currently listed in `getConfig("packages")`
`error.missing`	Boolean which decides whether an error is returned if the mapping file does not exist or not.
`returnPathOnly`	If set to TRUE only the file path is returned
`activecalc`	If set, this argument helps to define the first package within which the mapping has to be sought for. This happens via finding in which package the active calc function is located.

Value

the mapping as a data frame

Author(s)

Jan Philipp Dietrich

See Also

calcOutput, toolConvertMapping

Examples

```
head(toolGetMapping("regionmappingH12.csv", where = "madrat"))
```

`toolISOhistorical` *Tool: ISOhistorical*

Description

This function expects a MAgPIE object with ISO country codes in the spatial dimension. For this MAgPIE object the time of transition is calculated and for each the historic time filled by using the mapping stored as supplementary data in the madrat package. If you want to use a different mapping please specify it in the argument mapping

Usage

```
toolISOhistorical(
  m,
  mapping = NULL,
  additional_mapping = NULL,
  overwrite = FALSE,
  additional_weight = NULL
)
```

Arguments

`m`	MAgPIE object with ISO country codes in the spatial dimension
`mapping`	mapping of historical ISO countries to the standard ISO country list. For the default setting (mapping=NULL) the mapping stored as supplementary data in the madrat package is used. If provided as file the mapping needs to contain three columns "fromISO", "toISO" and "lastYear".
`additional_mapping`	vector or list of vectors to provide some specific mapping, first the old country code, second the new country code and last the last year of the old country, e.g. additional_mapping = c("TTT","TTX","y1111") or additional_mapping = list(c("TTT","TTX","y1111"),c("TTT","TTY","y1111"))
`overwrite`	if there are already historical data in the data source for years that are calculated in this function they will not be overwritten by default. To overwrite all data (e.g. if there are meaningless "0") choose overwrite=TRUE
`additional_weight`	optional weight to be used for regional disaggregation, if not provided, the values of m in the "lastYear" are used as weight

Value

A MAgPIE object with spatial entries for each country of the official ISO code country list. Historical time is filled up, old countries deleted

Author(s)

Lavinia Baumstark

`toolManualDownload` *Tool: ManualDownload*

Description

Support tool for the creation of download functions in cases where a fully automated data download is not an option (e.g. due to a missing API). The function can be used to print a step-by-step guide for the user how to manually retrieve the data and then asks for a (local) path where the data can be copied from.

Usage

```
toolManualDownload(
  instructions,
  intro = "Data must be downloaded manually",
  request = "Enter full path to the downloaded data:"
)
```

Arguments

`instructions`	Download instructions in form of a character vector describing how to manually retrieve the data.
`intro`	Introductory sentence to be shown first. Will not show up if set to NULL.
`request`	A prompt which should show up after the instructions to ask for the local download location.

Author(s)

Jan Philipp Dietrich

See Also

`downloadSource`

Examples

```
## Not run:
toolManualDownload(c("Log into website ABC",
                     "Download the data set XYZ"))

## End(Not run)
```

toolNAreplace *Tool: NA replace*

Description

Functions removes NAs, NaNs and infinite values in x and weight

Usage

```
toolNAreplace(x, weight = NULL, replaceby = 0, val.rm = NULL)
```

Arguments

x	data
weight	aggregation weight
replaceby	value which should be used instead of NA. Either a single value or a MAgPIE object which can be expanded to the size of x (either same size or with lower dimensionality).
val.rm	vector of values that should in addition be removed in x

Value

a list containing x and weight

Author(s)

Benjamin Bodirsky, Jan Philipp Dietrich

See Also

calcOutput

toolOrderCells *toolOrderCells*

Description

reorder numbered spatial units (cells, clusters) by number. Function will return the unmodified object, if the given subdimension does not exist or does not contain cell information.

Usage

```
toolOrderCells(x, dim = 1.2, na.rm = FALSE)
```

Arguments

x	magclass object that should be ordered
dim	subdimension which contains the cell information
na.rm	boolean deciding how to deal with non-integer information in cellular column. If FALSE, non-integer values will lead to a return of the unsorted object, if TRUE non-integer cells will be removed from the data set and the rest will get sorted

Value

ordered data in magclass format

Author(s)

Kristine Karstens, Jan Philipp Dietrich

---

toolSplitSubtype *Tool: SplitSubtype*

Description

This function can split a subtype string into smaller entities based on a given separator and check whether these entities exist in a reference list

Usage

```
toolSplitSubtype(subtype, components, sep = ":")
```

Arguments

subtype	A character string which can be split with the given separator into smaller entities
components	A named list with the same length as the subtype has entities. Names of the list are used as names of the entities while the content of each list element represents the allowed values of that given entity. If all values are allowed use NULL as entry.
sep	separator to be used for splitting

Value

A named list with the different entities of the given subtype

Author(s)

Jan Philipp Dietrich

Examples

```
toolSplitSubtype("mymodel:myversion:myworld", list(model=c("mymodel","notmymodel"),
                                                   version=c("myversion","42"),
                                                   world="myworld"))
```

---

toolstartmessage *Tool: Start message*

Description

This function writes a process start message (what function was called with which arguments) and stores the current time, so the corresponding call to toolendmessage can calculate the elapsed time.

Usage

```
toolstartmessage(functionName, argumentValues, level = NULL)
```

Arguments

functionName	The name of the calling function as a string.
argumentValues	A list of the evaluated arguments of the calling function.
level	This argument allows to establish a hierarchy of print statements. The hierarchy is preserved for the next vcat executions. Currently this setting can have 4 states: NULL (nothing will be changed), 0 (reset hierarchies), "+" (increase hierarchy level by 1) and "-" (decrease hierarchy level by 1).

Value

A list containing diagnostic information required by toolendmessage.

Author(s)

Jan Philipp Dietrich, Pascal Sauer

See Also

toolendmessage, vcat

Examples

```
innerFunction <- function() {
 startinfo <- madrat:::toolstartmessage("innerFunction", list(argumentsToPrint = 123), "+")
  vcat(1, "inner")
  madrat:::toolendmessage(startinfo, "-")
}
outerFunction <- function() {
  startinfo <- madrat:::toolstartmessage("outerFunction", list(), "+")
```

```
  vcat(1, "outer")
  innerFunction()
  madrat:::toolendmessage(startinfo, "-")
}
outerFunction()
```

`toolSubtypeSelect` *Tool: SubtypeSelect*

Description

This function is a support function for the selection of a subtype in a readX function. In addition to the subtype selection it also performs some consistency checks.

Usage

```
toolSubtypeSelect(subtype, files)
```

Arguments

`subtype`	A chosen subtype (character)
`files`	A named vector or list. The names of the vector correspond to the allowed subtypes and the content of the vector are the corresponding file names.

Value

The file name corresponding to the given subtype

Author(s)

Jan Philipp Dietrich

See Also

`readSource`

Examples

```
files <-  c(protection="protection.csv",
              production="production.csv",
              extent="forest_extent.csv")
toolSubtypeSelect("extent",files)
```

toolTimeAverage *toolTimeAverage*

Description

average over time given an averaging range. Only works for data with equidistant time steps!

Usage

```
toolTimeAverage(x, averaging_range = NULL, cut = TRUE, annual = NULL)
```

Arguments

x	magclass object that should be averaged with equidistant time steps
averaging_range	number of time steps to average
cut	if TRUE, all time steps at the start and end that can not be averaged correctly, will be removed if FALSE, time steps at the start and end will be averaged with high weights for start and end points
annual	deprecated. Please don't use it!

Value

the averaged data in magclass format

Author(s)

Kristine Karstens, Jan Philipp Dietrich

toolTimeSpline *toolTimeSpline*

Description

Smoothing a data set by replacing its values by its spline approximation using the given degrees of freedom.

Usage

```
toolTimeSpline(x, dof = NULL)
```

Arguments

x	magclass object that should be smoothed via a spline approximation
dof	degrees of freedom per 100 years (similiar to an average range), is a proxy for the smoothness of the spline (smaller values = smoother)

Value

approximated data in magclass format

Author(s)

Kristine Karstens, Felicitas Beier

`toolXlargest` *toolXlargest*

Description

Selects the countries with the highest values in a magpie object

Usage

```
toolXlargest(x, range = 1:20, years = NULL, elements = NULL, ...)
```

Arguments

`x`	magclass object that shall be used for ranking
`range`	the position of the countries in the top X which should be returned.
`years`	range of years that shall be summed for ranking. If NULL, the sum of all years is used.
`elements`	range of elements that shall be summed for ranking. If NULL, all elements are used.
`...`	further parameters will be handed on to calcOutput function type.

Value

vector with ISO country codes

Author(s)

Benjamin Leon Bodirsky, Jan Philipp Dietrich

Examples

```
toolXlargest(magclass::maxample("pop"), range = 1:3)
```

vcat *Tool: Verbosity Cat*

Description

Function which returns information based on the verbosity setting

Usage

```
vcat(
  verbosity,
  ...,
  level = NULL,
  fill = TRUE,
  show_prefix = TRUE,
  logOnly = FALSE
)
```

Arguments

verbosity	The lowest verbosity level for which this message should be shown (verbosity = -1 means no information at all, 0 = only warnings, 1 = warnings and execution information, 2 = full information). If the verbosity is set to 0 the message is written as warning, if the verbosity is set higher than 0 it is written as a normal cat message.
...	The message to be shown
level	This argument allows to establish a hierarchy of print statements. The hierarchy is preserved for the next vcat executions. Currently this setting can have 4 states: NULL (nothing will be changed), 0 (reset hierarchies), "+" (increase hierarchy level by 1) and "-" (decrease hierarchy level by 1).
fill	a logical or (positive) numeric controlling how the output is broken into successive lines. If FALSE (default), only newlines created explicitly by "\n" are printed. Otherwise, the output is broken into lines with print width equal to the option width if fill is TRUE, or the value of fill if this is numeric. Non-positive fill values are ignored, with a warning.
show_prefix	a logical defining whether a content specific prefix (e.g. "NOTE") should be shown in front of the message or not. If prefix is not shown it will also not show up in official statistics.
logOnly	option to only log warnings and error message without creating warnings or errors (expert use only).

Author(s)

Jan Philipp Dietrich

See Also

readSource

Examples

```
## Not run:
vcat(2, "Hello world!")

## End(Not run)
```

`visualizeDependencies` *visualizeDependencies*

Description

Creates a graphical visualization of dependencies between functions in the mr-universe.

Usage

```
visualizeDependencies(
  ...,
  direction = "both",
  order = 2,
  filter = NULL,
  packages = getConfig("packages"),
  filename = NULL
)
```

Arguments

`...`	function(s) to be analyzed
`direction`	Character string, either “in”, “out” or "both". If “in” all sources feeding into the function are listed. If “out” consumer of the function are listed. If “both” the union of "in" and "out" is returned.
`order`	order of dependencies. Order 1 would be only functions directly called from (in case of direction "in") or directly calling (in case of direction "out") are shown. Order 2 will also show direct dependencies of the order 1 dependencies, order 3 also the direct dependencies from order 2 dependencies, etc.
`filter`	regular expression to describe elements which should be excluded from visualization (e.g. "^tool" to exclude all tool functions)
`packages`	packages to use when searching dependencies
`filename`	If a filename is provided, the resulting graph will be saved

Author(s)

Debbora Leip, Jan Philipp Dietrich

See Also

getDependencies, getMadratGraph, getMadratInfo

---

withMadratLogging *Tool: withMadratLogging*

Description

Function will activate madrat logging facilities for all code provided to this function. This means that message, warning and stop calls will also report to the madrat log output

Usage

```
withMadratLogging(expr)
```

Arguments

expr	expression to be evaluated.

Author(s)

Jan Philipp Dietrich

See Also

vcat

Examples

```
## Not run:
madrat:::withMadratLogging(message("Hello world!"))

## End(Not run)
```

Index