Package ‘gms’

June 29, 2024

Type Package
Title 'GAMS' Modularization Support Package
Version 0.31.2
Date 2024-06-27
Description
A collection of tools to create, use and maintain modularized model code written in the modeling
language 'GAMS' (<https://www.gams.com/>). Out-of-the-box 'GAMS' does not come with support for modularized
model code. This package provides the tools necessary to convert a standard 'GAMS' model to a modularized one
by introducing a modularized code structure together with a naming convention which emu-
lates local
environments. In addition, this package provides tools to monitor the compli-
ance of the model code with
modular coding guidelines.
Imports dplyr, rlang, stringr, withr, yaml, filelock, stats
Suggests covr, curl, magclass, qgraph, testthat, callr
URL https://github.com/pik-piam/gms,
https://doi.org/10.5281/zenodo.4390032
BugReports https://github.com/pik-piam/gms/issues
License BSD_2_clause + file LICENSE
Encoding UTF-8
RoxygenNote 7.3.1
Config/Keywords tool
NeedsCompilation no
Author Jan Philipp Dietrich [aut, cre] (Potsdam Institute for Climate Impact
Research, <https://orcid.org/0000-0002-4309-6431>),
David Klein [aut] (Potsdam Institute for Climate Impact Research),
Anastasis Giannousakis [aut],
Felicitas Beier [aut] (Potsdam Institute for Climate Impact Research),
Johannes Koch [aut] (Potsdam Institute for Climate Impact Research),
Lavinia Baumstark [aut] (Potsdam Institute for Climate Impact Research),
Mika Pflüger [aut],
Oliver Richters [aut] (Potsdam Institute for Climate Impact Research)

Maintainer  Jan Philipp Dietrich <dietrich@pik-potsdam.de>
Repository  CRAN
Date/Publication  2024-06-29 06:50:10 UTC

Contents

gms-package . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
checkAppearance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
checkDescription . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
checkNoTabs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
checkSwitchAppearance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
check_config . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
chooseFromList . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
codeCheck . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
codeExtract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
convert.modularGAMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
copy_input . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
delete_olddata . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
download_distribute . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
download_unpack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
fulldataOutput . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
GAMScodeFilter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
getfiledestinations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
getLine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
getModules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
get_info . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
interfaceplot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
is.modularGAMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
loadConfig . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
model_dependencies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
model_lock . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
module.skeleton . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
modules_interfaceplot . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
path . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
publish_data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
readDeclarations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
readDefaultConfig . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
readFileOrVector . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
readParameterValues . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
readSetglobals . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
readSettings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
read_yaml_header . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
replace_in_file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
saveConfig . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
gms-package

gms: 'GAMS' Modularization Support Package

Description

A collection of tools to create, use and maintain modularized model code written in the modeling language 'GAMS' (https://www.gams.com/). Out-of-the-box 'GAMS' does not come with support for modularized model code. This package provides the tools necessary to convert a standard 'GAMS' model to a modularized one by introducing a modularized code structure together with a naming convention which emulates local environments. In addition, this package provides tools to monitor the compliance of the model code with modular coding guidelines.

Author(s)

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

Authors:

• David Klein (Potsdam Institute for Climate Impact Research)
• Anastasis Giannousakis
• Felicitas Beier (Potsdam Institute for Climate Impact Research)
• Johannes Koch (Potsdam Institute for Climate Impact Research)
• Lavinia Baumstark (Potsdam Institute for Climate Impact Research)
• Mika Pflüger
• Oliver Richters (Potsdam Institute for Climate Impact Research)

See Also

Useful links:

• https://github.com/pik-piam/gms
• doi: 10.5281/zenodo.4390032
• Report bugs at https://github.com/pik-piam/gms/issues
checkAppearance

Description

Checks for all declared objects in which parts of the model they appear and calculates the type of each object (core object, interface object, module object of module xy,...)

Usage

checkAppearance(x, capitalExclusionList = NULL)

Arguments

x
A code list as returned by codeExtract
capitalExclusionList
A vector of names that should be ignored when checking for unified capitalization of variables

Value

A list with four elements: appearance, setappearance, type and warnings. Appearance is a matrix containing values which indicate whether an object appears in a part of the code or not (e.g. indicates whether "vm_example" appears in realization "on" of module "test" or not.). 0 means that it does not appear, 1 means that it appears in the code and 2 means that it appears in the not_used.txt. setappearance contains the same information but for sets instead of other objects. Type is a vector containing the type of each object (excluding sets). And warnings contains a list of warnings created during that process.

Author(s)

Jan Philipp Dietrich

See Also

codeCheck, readDeclarations
checkDescription

Description
Checks whether all Declarations of a GAMS code come with a Description, throws out a warning in case of a missing description.

Usage
checkDescription(x, w = NULL)

Arguments
- x: GAMS declarations matrix as returned by readDeclarations
- w: a vector of warnings the warnings should be added to readDeclarations

Value
vector of warnings

Author(s)
Jan Philipp Dietrich

See Also
codeCheck

checkNoTabs

Description
Check all files (also in subdirectories) matching the given pattern for tabs. Will throw an error with a list of files where tabs were found if any.

Usage
checkNoTabs(pattern, exclude = NULL, excludeFolders = NULL)

Arguments
- pattern: A regular expression. Only files matching this pattern will be checked for tabs.
- exclude: A regular expression. Files matching this pattern will never be checked.
- excludeFolders: Paths to folders that should not be checked.
checkSwitchAppearance

Value

Invisibly, the list of files that were checked.

Author(s)

Pascal Sauer

Examples

```r
## Not run:
gms::checkNoTabs(pattern = \."(R|profile|gms|cfg|bib)$", excludeFolders = c("output", "renv", ".git"))
gms::checkNoTabs(utils::glob2rx("*.R"))
## End(Not run)
```

Description

Checks for all declared switches in which parts of the model they appear and calculates the type of each object (core object, interface object, module object of module xy,...)

Usage

`checkSwitchAppearance(code)`

Arguments

code Model code returned by `codeExtract`

Value

A list with three elements: switches, appearance and type. Switches is a vector containing all switches. The names of the vector contain the information where the switch is set. Appearance is a matrix containing values which indicate whether an object appears in a part of the code or not (e.g. indicates whether "vm_example" appears in realization "on" of module "test" or not.). 0 means that it does not appear, 1 means that it appears in the code and 2 means that it appears in the not_used.txt. Type is a vector containing the type of each object.

Author(s)

Jan Philipp Dietrich

See Also

codeCheck, readDeclarations, codeExtract, checkAppearance
check_config

Check config

Description

Checks a model configuration file for consistency by comparing it to a reference config file and the given module structure of the model. The function will throw out an error if settings are missing in the config which exist in the reference config, or if settings are set in the config which do not exist in the reference config file or if a realization is chosen for a module which does not exist, not allowed setting combinations.

Usage

```r
check_config(
  icfg,
  reference_file = "config/default.cfg",
  modulepath = "modules/",
  settings_config = NULL,
  extras = NULL,
  saveCheck = FALSE
)
```

Arguments

- `icfg`: Input config which should be checked for consistency (either as the config itself or as a file path linking to the config)
- `reference_file`: Reference config which is having the right format (either as the config itself or as a file path linking to the config)
- `modulepath`: The path where the modules are stored. If set to NULL the corresponding module check is deactivated.
- `settings_config`: path where the table of possible setting combinations is stored, if NULL it is ignored
- `extras`: vector of setting names that are allowed to appear in the input config even if they are missing in the reference config. That can be useful to allow for additional settings/information which does not necessarily have to exist
- `saveCheck`: additional check which makes sure that saving and loading it to/from YAML format does not change its contents

Value

The checked config as a config list ready for further usage.

Author(s)

Jan Philipp Dietrich, Lavinia Baumstark
chooseFromList

See Also
getModules

chooseFromList  chooseFromList

Description
Allows the user to select single or multiple items from a list. Entries can be selected based on individual choice, groups, regex or all.

Usage
chooseFromList(
  theList,
  type = "items",
  userinfo = NULL,
  addAllPattern = TRUE,
  returnBoolean = FALSE,
  multiple = TRUE,
  userInput = FALSE,
  errorMessage = NULL
)

Arguments
  theList  list or character vector to be selected from, names can specify groups
  type  string in plural shown to user to understand what they have to choose
  userinfo  string printed to the user before choosing
  addAllPattern  boolean whether 'all' and 'Search by pattern' options are added
  returnBoolean  TRUE: returns array with dimension of theList with FALSE and TRUE, which erases the order in which entries were selected FALSE: returns selected entries of theList, conserving the order in which entries were selected
  multiple  TRUE: allows to select multiple entries. FALSE: no
  userInput  string provided by the user. If not supplied, user is asked (mainly for testing)
  errorMessage  string used internally to tell the user before retrying that a selection does not make sense

Value
list or character vector, either a boolean with same length as theList or only the selected items.

Author(s)
Oliver Richters
**Examples**

```r
## Not run:
chooseFromList(
    theList = c(Letter = "A", Letter = "B", Number = "1", Number = "2"),
    type = "characters",
    userinfo = "Please don't select B, it hurts.",
    returnBoolean = FALSE,
    multiple = TRUE)

## End(Not run)
```

**Description**

Checks GAMS code for consistency. Throws out warnings if something is wrong in the code and returns a list containing the interfaces of each module of the code.

**Usage**

```r
codeCheck(
    path = ".",
    modulepath = "modules",
    core_files = c("core/*.gms", "main.gms"),
    returnDebug = FALSE,
    interactive = FALSE,
    test_switches = TRUE,
    strict = FALSE,
    details = FALSE
)
```

**Arguments**

- **path**: path of the main folder of the model
- **modulepath**: path to the module folder relative to "path"
- **core_files**: list of files that belong to the core (wildcard expansion is supported)
- **returnDebug**: If TRUE additional information will be returned useful for debugging the codeCheck function
- **interactive**: activates an interactive developer mode in which some of the warnings can be fixed interactively.
- **test_switches**: (boolean) Should realization switches in model core be tested for completeness? Usually set to TRUE but should be set to FALSE for standalone models only using a subset of existing modules
### strict
(boolea) test strictness. If set to TRUE warnings from codeCheck will stop calculations at the end of the analysis. Useful to enforce clean code.

### details
(boolea) If activated the function will return more detailed output. Besides interface information it will provide a table containing all declarations in the code, an appearance table listing the appearance of all objects in the code and information about the existing modules. The format is list(interfaceInfo,declarations,appearance,modulesInfo). This setting will be ignored when returnDebug is set to TRUE.

**Details**
Additional settings can be provided via a yaml file ".codeCheck" in the main folder of the model. Currently supported settings are: - capitalExclusionList: a list of names that should be ignored when checking for unified capitalization of variables

**Value**
A list of all modules containing the interfaces for each module. Or more detailed output if either details or returnDebug is set to TRUE.

**Author(s)**
Jan Philipp Dietrich

**See Also**
codeExtract, readDeclarations

**Examples**

```r
# check code consistency of dummy model
codeCheck(system.file("dummymodel", package = "gms"))
```

#### codeExtract
codeExtract

**Description**
Returns aggregated and cleaned GAMS code together with declaration matrix

**Usage**
codeExtract(codeFiles, name)

**Arguments**
codeFiles A vector of file names of GAMS code files.
name A name indicating what collection of code files this is (e.g. module name)
Value

A list with two elements: code and declarations. Code contains the cleaned up gams code and declarations contains the declarations matrix as returned by readDeclarations

Author(s)

Jan Philipp Dietrich

See Also

codeCheck, readDeclarations

Description

Converts modular GAMS code from an older modular definition to the newest one

Usage

convert.modularGAMS(path = ".", modulepath = "modules/"

Arguments

path path to the main folder of the model
modulepath Module path within the model (relative to the model main folder)

Author(s)

Jan Philipp Dietrich

See Also

codeCheck
**copy_input**

**Description**

Function to copy input files to their destination folders

**Usage**

```r
copy_input(x, sourcepath, suffix = NULL, move = FALSE)
```

**Arguments**

- `x` : Filepath or data frame containing the mapping of files to be deleted
- `sourcepath` : Path to folder containing all input files
- `suffix` : suffix that might be part of input names that should be deleted
- `move` : If TRUE files will be moved instead of copied (default=FALSE)

**Author(s)**

Jan Philipp Dietrich, David Klein

---

**delete_olddata**

**Description**

Delete data provided in mapping

**Usage**

```r
delete_olddata(x)
```

**Arguments**

- `x` : Filepath or data frame containing the mapping of files to be deleted

**Author(s)**

Jan Philipp Dietrich, David Klein
download_distribute

Download and unpack compressed data from repositories

Description

Downloads a list of tgz files from a list of repos and unpacks them

Usage

download_distribute(
  files,
  repositories = list(`/p/projects/rd3mod/inputdata/output` = NULL),
  modelfolder = ".",
  additionalDelete = NULL,
  debug = FALSE,
  stopOnMissing = FALSE
)

Arguments

files a vector of files containing input data, if a file is no .tgz it will be downloaded and not unpacked

repositories a list of repositories (please pay attention to the list format!) in which the files should be searched for. Files will be searched in all repositories until found, always starting with the first repository in the list. The argument must have the format of a named list with the url of the repository as name and a corresponding list of options such as username or password to access the repository as value. If no options are required the value has to be NULL. (e.g. list("ftp://my_pw_protected_server.de/data"=list(user="me",password=12345), "http://free_server.de/dat"

modelfolder main model folder

additionalDelete information which additional data should be deleted before new data are downloaded and distributed

ddebug switch for debug mode with additional diagnostic information

stopOnMissing Boolean passed along to download_unpack() to stop if any file in files could not be downloaded. Off (FALSE) by default.

Value

Information about the download process in form of a data.frame with data sets as row names and repositories (where it was downloaded from) and corresponding md5sum as columns

Author(s)

Jan Philipp Dietrich, Lavinia Baumstark
**download_unpack**

*Download and unpack compressed data from repositories*

**Description**

Downloads a list of tgz files from a list of repos and unpacks them. If a file is no .tgz-file it will be only downloaded.

**Usage**

```r
download_unpack(
  input,
  targetdir = "input",
  repositories = NULL,
  debug = FALSE,
  unpack = TRUE,
  stopOnMissing = FALSE
)
```

**Arguments**

- `input` a vector of files to be downloaded or a cfg list with settings to be used (e.g. containing cfg$input, cfg$repositories). Settings in the config list will be overwritten by other arguments of this function if they are not set to NULL
- `targetdir` directory the files should be downloaded and extracted to
- `repositories` a list of repositories (please pay attention to the list format!) in which the files should be searched for. Files will be searched in all repositories until found, always starting with the first repository in the list. The argument must have the format of a named list with the url of the repository as name and a corresponding list of options such as username or password to access the repository as value. If no options are required the value has to be NULL. (e.g. list("ftp://my_pw_protected_server.de/data"=list(user="me",password=12345), "http://free_server.de/data"=NULL))
- `debug` switch for debug mode with additional diagnostic information
- `unpack` if switched off the source files are purley downloaded
- `stopOnMissing` Boolean indicating whether to stop if any file in files could not be downloaded. Off (FALSE) by default.

**Value**

Information about the download process in form of a data.frame with data sets as row names and repositories (where it was downloaded from) and corresponding md5sum as columns

**Author(s)**

Jan Philipp Dietrich
**Description**

Creates GAMS code which stores automatically the levels, bounds and marginals of all equations and variables in time depending parameters.

**Usage**

```r
gamsfulldataOutput(  
declarations_file = "declarations.gms",  
definitions_file = "postsolve.gms",  
warn = TRUE,  
types = c("level", "marginal"),  
ignore = ".dummy$",  
loopset = "t"
)
```

**Arguments**

- **declarations_file**
  A GAMS file containing declarations. The function will read declarations from here and add own declarations in an R environment as used by `replace_in_file` (used subject = OUTPUT DECLARATIONS)

- **definitions_file**
  A GAMS file which is executed after the solve statement but within the time step loop. Also here code will be added using `replace_in_file` with subject OUTPUT DEFINITIONS

- **warn**
  Decides whether a warning should be thrown out, if the declarations file does not exist.

- **types**
  Types of outputs that should be written to gdx file. Available types are level, marginal, upper and lower.

- **ignore**
  regular expression pattern for variables/equations which should be ignored by fulldataOutput

- **loopset**
  Set over which loop runs

**Author(s)**

Jan Philipp Dietrich, Felicitas Beier

**See Also**

- `readDeclarations`, `replace_in_file`
GAMScodeFilter

Description
Cleans GAMS code supplied from empty lines and comments.

Usage
GAMScodeFilter(x)

Arguments
x A vector with lines of GAMS code (as you get by reading the code with readLines)

Value
The cleaned GAMS code

Author(s)
Jan Philipp Dietrich

See Also
readDeclarations

Examples
GAMScodeFilter(c("", "*comment", "a=12;", "", "b=13;"))

getfiledestinations

Description
Create file2destination mapping based on information from the model, ignoring top-level directories listed in .gitignore.

Usage
getfiledestinations(path = ".", ignoreFolders = "renv")
getLine

Arguments

path main model folder
ignoreFolders folders to be ignored by the function, additionally to directories listed in .gitignore (by default only "renv").

Author(s)

Jan Philipp Dietrich, David Klein

getLine

Description

Get one line of user input regardless of whether running interactively or not (via Rscript). (base::readline does not wait for user input when running via Rscript.)

Usage

geline()

Value

The user input as a string.

getModules

Description

Extract module information of a GAMS model.

Usage

getModules(modulepath)

Arguments

modulepath The path where the modules are stored.

Value

A matrix containing the different modules with name, corresponding module number and corresponding realizations
Author(s)

Jan Philipp Dietrich

See Also

codeCheck

---

get_info

get_info

---

Description

Function to extract information from info.txt

Usage

get_info(file, grep_expression, sep, pattern = "", replacement = "")

Arguments

file            path to info.txt (including info.txt)
grep_expression String before the information that should be extracted
sep             Separator between grep_expression and information
pattern         Optional pattern to be replaced (default pattern = "")
replacement     Optional replacement (default replacement = "")

Author(s)

Jan Philipp Dietrich, David Klein

---

interfaceplot

interfaceplot

---

Description

Creates an interface plot of a modular model using qgraph and returns the interface information.
Usage

```r
interfaceplot(
  x = ".",
  modules_to_include = NULL,
  modules_to_exclude = NULL,
  links_to_include = NULL,
  links_to_exclude = NULL,
  items_to_include = NULL,
  items_to_exclude = NULL,
  items_to_display = NULL,
  default_groups = list(default1 = list(name = "core", nodes = "core", color = "black",
                                        shape = "rectangle"), default2 = list(name = "modules", nodes = NULL, color = "#6c9ebf", shape = "ellipse")),
  highlight_groups = NULL,
  max_length_node_names = NULL,
  add_nodeName_legend = FALSE,
  max_num_edge_labels = NULL,
  max_num_nodes_for_edge_labels = 30,
  ...
)
```

Arguments

- `x`: Either an interface list as returned by `codeCheck` or the path to the main folder of the model.
- `modules_to_include`: NULL (default value) or a vector of strings with names of modules to include, e.g. c("core", "macro"). If NULL all modules are included.
- `modules_to_exclude`: NULL (default value) or a vector of strings with names of modules to exclude, e.g. c("core"). If NULL no modules are excluded.
- `links_to_include`: NULL (default value) or list of lists with attributes "to" and "from", that each take a vector of module names, e.g. list(list(to="macro", from="core")). If NULL all links are included.
- `links_to_exclude`: NULL (default value) or list of lists with attributes "to" and "from", that each take a vector of module names, e.g. list(list(to="macro", from="core")). If NULL no links are excluded.
- `items_to_include`: NULL (default value) or a vector of strings with names of items to include, e.g. c("vm_cesIO", "pm_pvp"). Regex patterns can also be passed, e.g. c("(vl)p_m_.*"). If NULL all items are included.
- `items_to_exclude`: NULL (default value) or a vector of strings with names of items to exclude, e.g. c("vm_cesIO", "pm_pvp"). Regex patterns can also be passed, e.g. c("sm_.*"). If NULL no items are excluded.
items_to_display
NULL (default value) or a vector of strings with names of items to display, e.g.
c("vm_cesIO", "pm_pvp"). Regex patterns can also be passed, e.g. c(m\ S+). If NULL no items are displayed.

default_groups
List of lists with default group definitions. Defines the default formatting of the
interface plot. By default, there are two groups, see usage, a "core" group made
up of only the "core" module and a "modules" group made up of all the rest. If
a "core" module doesn't exist, then that group is simply ignored.

highlight_groups
NULL (default value) or a list of lists with highlight-group definitions. By defin-
ing highlight groups, additional/specialized formatting can be applied to select
modules. A group is defined by a list with the following attributes:

• name: a string with the group name. Will appear in legend.
• nodes: a vector of strings with module names.
• shape: a string with a valid qgraph shape.
• color: a string with a valid qgraph color.
• edges_to_highlight:
  – NULL = no edges are colored
  – "all" = edges starting from and ending at the highlight group’s nodes
    are colored
  – "incoming" = edges ending at the highlight group’s nodes are colored
  – "outgoing" = edges starting from the highlight group’s nodes are col-
    ored
  – "within" = only edges that departed from the highlight group’s nodes
    and end at them as well are colored
• edges_to_ignore:
  – NULL = no edges are ignored
  – "outside" edges that neither start from or end at any of the group’s nodes
    are ignored
  – "incoming" edges that do not depart from one of the group’s nodes are
    ignored
  – "outgoing" edges that do not arrive at one of the group’s nodes are
    ignored
  – "outgoing_to_no_return" edges that departing from nodes outside of
    the group and not ending at nodes within the group are ignored

An example: list(list(name = "highlight", nodes = "welfare", color =
"#ff8f00", shape = "ellipse", edges_to_highlight = "outgoing", edges_to_ignore = "outside").

max_length_node_names
NULL (default value) or an integer n giving the maximum number of characters
allowed in the node names. If not NULL, node names are truncated after n
characters, e.g. n=3: "example" -> "exa."

add_nodeName_legend
Logical (default FALSE) to add node names in legend, structured by group.
max_num_edge_labels

NULL (default value), an integer or the string "adjust". If NULL, all edge labels are displayed. If given an integer n, a maximum of n edge labels are shown. If set to "adjust", the number of edge labels displayed decreases with the number of nodes.

max_num_nodes_for_edge_labels

Integer, (default value = 30). The maximum number of nodes after which no edge labels are displayed.

... Optional arguments to qgraph.

Details

What modules (=nodes), links (=edges) and items (=what is passed along the edges) are taken into account when creating the plot can be fine-tuned with the "_include", "_exclude" arguments.

The "default"- and "highlight_groups" arguments control the formatting (and also the composition through "highlight_group$edges_to_ignore"). Groups in qgraph are a way of clustering nodes together. The default formatting of the plot is defined with the "default_groups" argument. On top of that additional groups can be defined with the "highlight_groups" argument.

The rest of the arguments are pretty self-explanatory. Just remember that qgraph arguments can be passed on as well! Useful ones include: fade=T/F, legend=T/F, legend.cex (size of the legend font), GLratio (graph/legend size ratio, edge.label.cex (size of the edge label font)).

Value

A tibble with the edge list and interface items.

Author(s)

Johannes Koch

See Also

codeCheck.qgraph

Description

Checks whether a folder seems to contain modular GAMS code or not.

Usage

is.modularGAMS(path = ".", version = FALSE, modulepath = "modules/"
loadConfig

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>path to the main folder of the model</td>
</tr>
<tr>
<td>version</td>
<td>if TRUE returns the version of the modular structure or FALSE, otherwise return a boolean indicating whether it is modular or not.</td>
</tr>
<tr>
<td>modulepath</td>
<td>Module path within the model (relative to the model main folder)</td>
</tr>
</tbody>
</table>

Author(s)

Jan Philipp Dietrich

See Also

codeCheck

Examples

is.modularGAMS(system.file("dummymodel",package="gms"))

Description

Load config in YAML format as written via saveConfig.

Usage

loadConfig(cfg)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cfg</td>
<td>Either a character string naming a file which contains the config or a character string containing the config as YAML code.</td>
</tr>
</tbody>
</table>

Details

To read in the default configuration (stored as R list in default.cfg or in a gams file), use readDefaultConfig instead.

Author(s)

Jan Philipp Dietrich

See Also

saveConfig
Examples

```r
cfg <- list(input = c(data1 = "blub.tgz", data2 = "bla.tgz"), mode = "default")
yml <- saveConfig(cfg)
loadConfig(yml)
```

---

**model_dependencies**  
*Function to detect R package dependencies*

**Description**

This function analyzes a model folder and all subfolders and searches for library and require statements.

**Usage**

```r
model_dependencies(mainfolder = ".")
```

**Arguments**

- `mainfolder`: main folder of the model to be analyzed

**Value**

A list of dependencies sorted by appearances

**Author(s)**

Jan Philipp Dietrich

---

**model_lock**  
*Model lock/unlock*

**Description**

Lock the model to coordinate multiple processes accessing the model. This is necessary if you change files within the model and have to avoid that another process reads from your half-written changes in the mean time. `model_lock` creates an exclusive lock on the given file and returns a lock id. When the lock id is handed to `model_unlock` the lock is dropped again, and another process can take the lock. The lock is also dropped when the lock id goes out of scope and is garbage collected in R or when the R process terminates or crashes. However, it is good practice to explicitly call `model_unlock` when you don't need the lock anymore. If you want to check for informational purposes if the model is currently locked by another process, you can use the `is_model_locked()` function. Note however that you should never use `is_model_locked()` to coordinate access to the model folder. The return value of `is_model_locked()` is only a snapshot at the time when the function is run. Note additionally that `is_model_locked()` checks if the model is locked by another process. If you currently hold the lock of the model yourself, it will return FALSE.
Usage

model_lock(folder = ".", file = ".lock", timeout1 = 12, timeout2 = NULL, check_interval = NULL, oncluster = NULL)
model_unlock(id, folder = NULL, file = NULL, oncluster = NULL)
is_model_locked(folder = ".", file = ".lock")

Arguments

folder path to the model folder
file file name of the lock file. Don’t use the lock file for anything else (i.e. don’t read or write or open the lock file).
timeout1 Time in hours to wait for other processes to free the lock before giving up and exiting with an error.
timeout2 deprecated setting which will be ignored.
check_interval deprecated setting which will be ignored.
oncluster deprecated setting which will be ignored.
id lock id as returned by model_lock.

Value

model_lock returns the lock id which is needed to identify the lock in model_unlock.

Author(s)

Jan Philipp Dietrich, David Klein, Mika Pflüger

See Also

check_config

Examples

#lock folder
id <- model_lock(tempdir())

#unlock folder
model_unlock(id)
Create a Module skeleton

Description

This function creates you a module skeleton which you can use to easily create your own modules.

Usage

```r
module.skeleton(
  number,
  name,
  types,
  modelpath = ".",
  modulepath = "modules/",
  includefile = "modules/include.gms",
  version = is.modularGAMS(modelpath, version = TRUE)
)
```

Arguments

- **number**: Number of your module, typically something between 0-99. Sorts the execution of your modules. Please use a number which is not used, yet.
- **name**: Name of your module (please choose a short name). If you want to extend an existing module (add a new realisation) use the name of the existing one.
- **types**: Vector of names for the different module types (e.g. "on" or "off"). If you want to extend an existing module (add a new realisation), put here the additional type(s)
- **modelpath**: Path of the MAgPIE version that should be updated (main folder).
- **modulepath**: Module path within MAgPIE (relative to the MAgPIE main folder)
- **inclusedefile**: Name and location of the file which includes all modules (relative to main folder)
- **version**: Version of the modular GAMS code structure (1 or 2)

Note

Module phases are automatically detected checking the main code of the model, but not checking code in modules. If you want to use additional phases which are only included within a module, you need to specify them manually by adding a comment into your gams code indicating that there is an additional phase. The syntax is "* !add_phase!: <phase>", e.g. "* !add_phase!: new_phase"

Author(s)

Jan Philipp Dietrich
Examples

```r
# copy dummymodel to temporary directory and add new module "bla"
file.copy(system.file("dummymodel", package="gms"), tempdir(), recursive = TRUE)
model  <- paste0(tempdir(), "/dummymodel")
module.skeleton(number="03", name="bla", types=c("on", "off"), modelpath=model)
```

Description

Function that applies `interfaceplot` for a whole model and all its modules.

Usage

```r
modules_interfaceplot(
  x = ".",
  modulepath = "modules",
  filetype = "png",
  targetfolder = NULL,
  writetable = TRUE,
  includeCore = FALSE,
  ...
)
```

Arguments

- `x` Either the object returned by `codeCheck` or the path to the main folder of the model.
- `modulepath` Path to the modules folder
- `filetype` Filetype that should be used (e.g. "png" or "pdf")
- `targetfolder` Folder outputs should be written to. If set to NULL outputs will be added to corresponding module folders
- `writetable` Logical deciding whether a csv containing the interfaces should be written as well.
- `includeCore` Logical to create plot for core or not, default FALSE.
- `...` Optional arguments to `interfaceplot`.

Value

A list with interface tables for each module

Author(s)

Jan Philipp Dietrich
path

See Also

codeCheck, interfaceplot

Description

Small function to build a consistent path-string based on folder, filename and filetype. The function makes sure that slashes and the dot for the file ending are set correctly (you can supply your folder name either with or without a trailing slash in it. It does not matter.

Usage

path(..., ftype = NULL)

Arguments

... the folders and the file name that should be pasted to a file/folder path
ftype file type

Value

A string containing the path combined of folder, filename and filetype

Author(s)

Jan Philipp Dietrich

publish_data Publish data in a repository

Description

Downloads a list of tgz files from a list of repos, merge them and publish it on another server

Usage

publish_data(
  input,
  name = NULL,
  target = Sys.getenv("PUBLISH_DATA_TARGET", unset = "."),
  ...
)
Arguments

- **input**: a vector of files to be downloaded or a cfg list with settings to be used (e.g. containing `cfg$input`, `cfg$repositories`). Settings in the config list will be overwritten by other arguments of this function if they are not set to NULL.
- **name**: name of the data to be published (will be used in as file name). If no name is given (default) source files will be published as is (separate tgz files with original name).
- **target**: target the data should be published in (format `user@server:/folder/path`) If a target vector, or targets separated by "|" are provided the user will be asked interactively where the file should be written to. By default it will look for target information in the environment variable `PUBLISH_DATA_TARGET`.
- **...**: further options provided to `download_unpack`

Author(s)

Jan Philipp Dietrich

See Also

- `download_unpack`, `tardir`

Description

Reads all declarations given in a GAMS code and returns them.

Usage

```r
readDeclarations(
  file,
  unlist = TRUE,
  types = c("scalar", "(positive |)variable", "parameter", "table", "equation", "set")
)
```

Arguments

- **file**: A gams file or a vector containing GAMS code.
- **unlist**: A logical indicating whether the output should be returned as a list separated by object type or a matrix.
- **types**: of declarations to be read.

Value

Either a list of declared objects or a matrix containing object name, the sets the object depends on and the description.
readDefaultConfig

**Author(s)**

Jan Philipp Dietrich

**See Also**

codeCheck

---

**readDefaultConfig**

---

**Description**

Reads the default configuration of the model. Uses default.cfg or main.cfg as the source as appropriate. To read a configuration from YAML format, use `loadConfig` instead.

**Usage**

`readDefaultConfig(path)`

**Arguments**

- **path**
  
  path of the main folder of the model

**Value**

A vector of parameter values and their names.

**Author(s)**

Mika Pflüger

**See Also**

loadConfig
**Description**

Input: a file name or a character vector. Output: a character vector of the contents of the file, or the character vector unchanged.

**Usage**

readFileOrVector(file)

**Arguments**

file 
A file name or a character vector.

**Value**

A character vector of the inputs of the file.

**Author(s)**

Mika Pflüger

**Description**

Read values of given parameters from a given GAMS code section.

**Usage**

readParameterValues(code, parameters)

**Arguments**

code 
A vector with GAMS code.

parameters 
A vector with GAMS parameter names

**Details**

The GAMS code section should contain statements of the form parameter = value; for all the given parameters.
**readSetglobals**

**Value**
A vector of values the parameters are set to with parameter names as names.

**Author(s)**
Mika Pflüger

**See Also**
readDeclarations

---

**Description**
Reads all setglobals given in a GAMS code and returns them.

**Usage**

```plaintext
readSetglobals(file)
```

**Arguments**

*file* A gams file or a vector containing GAMS code.

**Value**
A vector of values the setglobal variables are set to with setglobal variables as names.

**Author(s)**
Jan Philipp Dietrich

**See Also**
readDeclarations
### readSettings

**Description**
Reads all settings (parameters, globals, and their respective values) in the given GAMS code.

**Usage**

```plaintext
readSettings(file)
```

**Arguments**
- `file`: A gams file or a vector containing GAMS code.

**Value**
A vector of parameter values and their names.

**Author(s)**
Mika Pflüger

**See Also**
- `readDeclarations`

### read_yaml_header

**Description**
Reads header written in yaml format from a file.

**Usage**

```plaintext
read_yaml_header(file, n = 20)
```

**Arguments**
- `file`: path to the file which contains the YAML header
- `n`: Number of lines to be read (header must be part of these lines in order to be read)

**Value**
A list containing the read in information
replace_in_file

**Author(s)**
Jan Philipp Dietrich

---

**replace_in_file**  
*Replace in File*

**Description**

Function to replace a marked paragraph in a text file. Paragraph has to be marked in the text file with an initial "##### R SECTION START (SUBJECT) ######" and "##### R SECTION END (SUBJECT) ######" as ending. The number of # symbols can be chosen by the user, but there has to be at least one at the beginning and one at the end. Furthermore it is allowed to add further symbols at the beginning or the end of the line. "SUBJECT" is chosen by the user and is used for identification, if a text file has more than one R section.

**Usage**

```r
replace_in_file(
  file,
  content,
  subject = "CODE",  
  add = FALSE,
  addfile = FALSE,
  comment = "#"
)
```

**Arguments**

- **file**: a connection object or a character string describing the file, that should be manipulated.
- **content**: the content that should be used as replacement stored as a vector of strings. Each vector component will be written as a line.
- **subject**: A string used for identification of a paragraph.
- **add**: Determines behavior when marking is missing in the code. add=FALSE will throw out an error, if the marking is missing, add="top" will add the markings automatically at the beginning of the file, add="bottom" or add=TRUE will do the same but at the end of the file.
- **addfile**: Determines the behavior when the file does not exist. If addfile=TRUE, file will be created when missing.
- **comment**: Symbol which is used to indicate a comment in the language the file is written that should be manipulated. Only relevant if add or addfile are used.

**Author(s)**
Jan Philipp Dietrich
**saveConfig**  
*Save Config*

**Description**
Saves config in YAML format so that it can be read in again via `loadConfig`.

**Usage**
```r
saveConfig(cfg, file = NULL)
```

**Arguments**
- `cfg`  
  Input config which should be saved
- `file`  
  A character string naming a file. If set to NULL the YAML output will be returned by the function.

**Author(s)**
Jan Philipp Dietrich

**See Also**
- `loadConfig`

**Examples**
```r
cfg <- list(input = c(data1 = "blub.tgz", data2 = "bla.tgz"), mode = "default")
saveConfig(cfg)
```

---

**selectScript**  
*selectScript*

**Description**
Functions which allows for interactive selection of scripts/files.

**Usage**
```r
selectScript(folder = ".", ending = "R")
```

**Arguments**
- `folder`  
  Folder in which the files/scripts are located which should be selected from.
- `ending`  
  File ending of the files to be selected (without dot)
**Value**

A vector of paths to files selected by the user

**Author(s)**

Jan Philipp Dietrich

---

**Description**

`setScenario` is adapting a given config to a predefined scenario, meaning that all settings which are fixed for the given scenario are written to the config. Settings not defined by the scenario remain unchanged.

**Usage**

```r
setScenario(cfg, scenario, scenario_config = "config/scenario_config.csv")
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cfg</code></td>
<td>Input config which should be adapted to the given scenario</td>
</tr>
<tr>
<td><code>scenario</code></td>
<td>name of scenario (e.g. &quot;SSP2&quot;). Can also be a vector of scenarios. In this case scenario settings are applied in the given order</td>
</tr>
<tr>
<td><code>scenario_config</code></td>
<td>The path where the scenario config table is stored.</td>
</tr>
</tbody>
</table>

**Value**

The updated config as a config list ready for further usage.

**Note**

The scenario config table is a table which contains as columns the different scenarios and as rows the different settings. Empty entries for a given scenario-setting combination indicate that this setting is not defined by the scenario and should not be changed by `set Scenario`!

**Author(s)**

Jan Philipp Dietrich, Anastasis Giannousakis

**See Also**

`check_config`, `getModules`
settingsCheck  

Description
Checks GAMS setglobals in code for consistency. Creates a warning if a setglobal command for an existing module is missing or a module is set to a realization which does not exist.

Usage
settingsCheck(path = ".", modulepath = "modules", fileName = "main.gms")

Arguments
- path: path of the main folder of the model
- modulepath: path to the module folder relative to "path"
- fileName: name of the file containing setglobals, relative to "path"

Value
Nothing is returned.

Author(s)
Jan Philipp Dietrich

See Also
codeCheck

singleGAMSfile  

Merge GAMS code into single file

Description
This function merges GAMS code which is distributed over several files into a single GAMS file. Optionally, it also embeds R scripts into the single GAMS file.

Usage
singleGAMSfile(
    modelpath = ".",
    mainfile = "main.gms",
    output = "full.gms",
    embedRScripts = FALSE
)
**Arguments**

- **modelpath**: The path where the model is stored
- **mainfile**: The path to the main gams file (relative to the model path)
- **output**: Name of the single output GAMS file.
- **embedRScripts**: If TRUE, R scripts called by GAMS via Execute are also embedded. Default FALSE

**Author(s)**

Jan Philipp Dietrich, Anastasis Giannousakis

**Examples**

```r
# copy dummymodel create single gms file out of it
dummymodel <- file.copy(system.file("dummymodel", package = "gms"), tempdir(), recursive = TRUE)
model <- paste0(tempdir(), "/dummymodel")
singlefile <- paste0(tempdir(), "/full.gms")
singleGAMSfile(modelpath = model, output = singlefile)
```

---

**tardir**

Create tgz archive from directory

**Description**

Creates a tgz from all files in a directory

**Usage**

tardir(dir = ".", tarfile = "data.tgz")

**Arguments**

- **dir**: directory from which the tar file should be generated
- **tarfile**: name of the archive the data should be written to (tgz file)

**Author(s)**

Jan Philipp Dietrich

**Examples**

```r
# copy dummymodel to temporary directory and compress it
dummymodel <- file.copy(system.file("dummymodel", package = "gms"), tempdir(), recursive = TRUE)
model <- paste0(tempdir(), "/dummymodel")
archive <- paste0(tempdir(), "/dummymodel.tgz")
tardir(model, archive)
```
**updateInterfaceMapping**

*updateInterfaceMapping*

**Description**

Function to update the mapping between interfaces and their origin modules.

**Usage**

`updateInterfaceMapping(path = ".", modulepath = "modules")`

**Arguments**

- `path`  
  path of the main folder of the model
- `modulepath`  
  path to the module folder relative to "path"

**Author(s)**

Jan Philipp Dietrich

**See Also**

codeCheck

---

**update_fulldataOutput**

*update_fulldataOutput*

**Description**

Creates GAMS code which stores automatically the levels and marginals of all equations and variables in time depending parameters.

**Usage**

`update_fulldataOutput(
    modelpath = ".",
    modulepath = "modules",
    corepath = "core",
    loopset = "t"
)`
update_modules_embedding

Arguments

- `modelpath`: Path of the Model version that should be updated (main folder).
- `modulepath`: Module path within the model (relative to the model main folder).
- `corepath`: Core path within the model (relative to the model main folder).
- `loopset`: Set over which loop runs.

Author(s)

Jan Philipp Dietrich, Felicitas Beier

See Also

- `fulldataOutput.replace_in_file`

Description

A function that updates in the GAMS code all include commands which are related to Modules. The function automatically checks which modules exist and which files in these modules exist and creates the corresponding include commands in GAMS.

Usage

```gams
update_modules_embedding(
    modelpath = ".", 
    modulepath = "modules/", 
    includefile = "modules/include.gms", 
    verbose = FALSE 
)
```

Arguments

- `modelpath`: Path to the model that should be updated (main folder).
- `modulepath`: Module path within the model (relative to the model main folder).
- `includefile`: Name and location of the file which includes all modules (relative to main folder).
- `verbose`: Defines whether additional information should be printed or not.

Note

Module phases are automatically detected checking the main code of the model, but not checking code in modules. If you want to use additional phases which are only included within a module, you need to specify them manually by adding a comment into your gams code indicating that there is an additional phase. The syntax is "* !add_phase!: <phase>", e.g. "* !add_phase!: new_phase"
Author(s)

Jan Philipp Dietrich

Examples

# copy dummymodel to temporary directory and update module embedding
file.copy(system.file("dummymodel", package="gms"), tempdir(), recursive = TRUE)
model <- paste0(tempdir(),"/dummymodel")
update_modules_embedding(model)

writeSets

Description

Function to write a set declaration in GAMS syntax given either a character vector or a data.frame
(latter to represent a mapping set)

Usage

writeSets(sets, file = NULL)

Arguments

sets a list of sets where every set is another list consisting of three entries: name (set
name as character), desc (set description as character) and items (set elements,
either as character vector or data.frame)

file Name of the file the set declarations should be written to. If NULL the result
will just be returned by the function. The file must exist and contain a line of
the form "##### R SECTION START (SETS) #####" followed by a line of the
form "##### R SECTION END (SETS) #####" to indicate the part of the file that
should be replaced with the set declaration!

Value

set declaration in GAMS syntax

Author(s)

Jan Philipp Dietrich, Florian Humpenöder
Examples

countries <- c("DEU", "FRA", "ENG", "ITA", "NLD", "POL")
map <- data.frame(region = rep(c("REG1","REG2"), 3), countries = countries)
sets <- list(list(name = "countries",
               desc = "list of countries",
               items = countries),
             list(name = "country2region",
               desc = "mapping between countries and regions",
               items = map))
cat(writeSets(sets), sep="\n")
Index

check_config, 7, 24, 35
checkAppearance, 4, 6
checkDescription, 5
checkNoTabs, 5
checkSwitchAppearance, 6
chooseFromList, 8
codeCheck, 4–6, 9, 11, 18, 19, 21, 22, 26, 27, 29, 36, 38
codeExtract, 4, 6, 10, 10
convert.modularGAMS, 11
copy_input, 12
delete_olddata, 12
download_distribute, 13
download_unpack, 14, 28
download_unpack(), 13
fulldataOutput, 15, 39
GAMSCodeFilter, 16
get_info, 18
getfiledestinations, 16
getLine, 17
getModules, 8, 17, 35
gms (gms-package), 3
gms-package, 3
interfaceplot, 18, 26, 27
is.modularGAMS, 21
is_model_locked (model_lock), 23
loadConfig, 22, 29, 34
model_dependencies, 23
model_lock, 23
model_unlock (model_lock), 23
module.skeleton, 25
modules_interfaceplot, 26
path, 27
publish_data, 27
qgraph, 18, 21
read_yaml_header, 32
readDeclarations, 4–6, 10, 11, 15, 16, 28, 31, 32
readDefaultConfig, 22, 29
readFileOrVector, 30
readParameterValues, 30
readSetglobals, 31
readSettings, 32
replace_in_file, 15, 33, 39
saveConfig, 22, 34
selectScript, 34
setScenario, 35
settingsCheck, 36
singleGAMSfile, 36
tardir, 28, 37
update_fulldataOutput, 38
update_modules_embedding, 39
updateInterfaceMapping, 38
writeSets, 40