Package 'gms'

October 13, 2022

Type Package

Title 'GAMS' Modularization Support Package

Version 0.4.0

Date 2020-06-17

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.

Imports dplyr, rlang, stringr, yaml

Suggests curl, magclass, qgraph, testthat

URL https://github.com/pik-piam/gms

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

License BSD_2_clause + file LICENSE

LazyData no

Encoding UTF-8

RoxygenNote 7.1.0

NeedsCompilation no

Author Jan Philipp Dietrich [aut, cre],
David Klein [aut],
Anastasis Giannousakis [aut],
Felicitas Beier [aut],
Johannes Koch [aut],
Lavinia Baumstark [aut]

Maintainer Jan Philipp Dietrich <dietrich@pik-potsdam.de>

Repository CRAN

Date/Publication 2020-07-01 15:00:06 UTC

R topics documented:

checkAppearance *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)
```

Arguments

x	A code list as returned by codeExtract

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 (exluding sets). And warnings contains a list of warnings created during that process.

Author(s)

Jan Philipp Dietrich

See Also

codeCheck,readDeclarations

checkDescription *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
w	a vector of warnings the warnings should be added to readDeclarations

Value

vector of warnings

Author(s)

Jan Philipp Dietrich

See Also

codeCheck

---

checkSwitchAppearance *checkSwitchAppearance*

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, of 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

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

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

Value

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

Author(s)

Jan Philipp Dietrich, Lavinia Baumstark

See Also

getModules

codeCheck *codeCheck*

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

```
codeCheck(
  path = ".",
  modulepath = "modules",
  core_files = c("core/*.gms", "main.gms"),
  debug = 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)
debug	If TRUE additional information will be returned usefule 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 completness? Usually set to TRUE but should be set to FALSE for standalone models only using a subset of existing modules
strict	(boolean) test strictness. If set to TRUE warnings from codeCheck will stop calculations at the end of the analysis. Useful to enforce clean code.
details	(boolean) 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 debug is set to TRUE.

Value

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

Author(s)

Jan Philipp Dietrich

See Also

codeExtract,readDeclarations

Examples

```
# 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

convert.modularGAMS *convert.modularGAMS*

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 *copy_input*

Description

Function to copy input files to their destination folders

Usage

```
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` *delete_olddata*

Description

Delete data provided in mapping

Usage

```
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
)
```

Arguments

`files`	a vector of files containing input data
`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 additonal data should be deleted before new data are downloaded and distributed
debug	switch for debug mode with additional diagnostic information

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

Usage

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

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/dat
debug	switch for debug mode with additional diagnostic information
unpack	if switched off the source files are purley downloaded

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

`fulldataOutput` *fulldataOutput*

Description

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

Usage

```
fulldataOutput(
  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 *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` *getfiledestinations*

Description

Create file2destination mapping based on information from the model

Usage

```
getfiledestinations()
```

Author(s)

Jan Philipp Dietrich, David Klein

`getModules` *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

```
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("(v|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 defining highligh groups, additional/specalized 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 colored
  - "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 lables 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 qqgraph 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

`is.modularGAMS` *is.modularGAMS*

Description

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

Usage

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

Arguments

`path`	path to the main folder of the model
`version`	if TRUE returns the version of the modular structure or FALSE, otherwise returns a boolean indicating whether it is modular or not.
`modulepath`	Module path within the model (relative to the model main folder)

Author(s)

Jan Philipp Dietrich

See Also

codeCheck

Examples

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

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

```
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

Functions that indicate whether a model folder is currently locked by another process or not. This helps to prevent unintended interactions between processes.

Usage

```
model_lock(folder=".", file=".lock", timeout1=NULL, timeout2=NULL,
check_interval=1, oncluster=TRUE)
model_unlock(id,folder=".",file=".lock",oncluster=TRUE)
```

Arguments

folder	model folder
file	file name of the lock file containing the process queue
timeout1	Time in hours the top process in the queue is allowed to run before the current process is stopped.
timeout2	Time in hours the processed is allowed to wait in the queue before it is stopped

check_interval	Time in seconds between checking the current position in the queue.
oncluster	a logical indicating whether the script is run on cluster or not. On windows a lock file is created, which does not prevent simulatneous access to the model. On the cluster the system command 'mkdir' is used to prevent simultaneous access. This atomicity of check-and-create is ensured at the operating system kernel level.
id	process id as returned by model_lock.

Value

model_lock returns the process id which is needed (only on Windows) to identify the process in model_unlock.

Author(s)

Jan Philipp Dietrich, David Klein

See Also

check_config

Examples

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

#unlock folder
model_unlock(id,tempdir())
```

---

module.skeleton *Create a Module skeleton*

Description

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

Usage

```
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)
includefile	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

```
# 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)
```

---

modules_interfaceplot *modules_interfaceplot*

Description

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

Usage

```
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

See Also

codeCheck,interfaceplot

---

path *path*

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 tailing 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

---

readDeclarations *readDeclarations*

Description

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

Usage

```
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.

Author(s)

Jan Philipp Dietrich

See Also

codeCheck

readSetglobals *readSetglobals*

Description

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

Usage

```
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

read_yaml_header *read_yaml_header*

Description

Reads header written in yaml format from a file

Usage

```
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

Author(s)

Jan Philipp Dietrich

`replace_in_file` *Replace in File*

Description

Function to replace a marked paragaph 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

```
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

selectScript *selectScript*

Description

Functions which allows for interactive selection of scripts/files.

Usage

```
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

setScenario *setScenario*

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

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

Arguments

cfg	Input config which should be adapted to the given scenario
scenario	name of scenario (e.g. "SSP2"). Can also be a vector of scenarios. In this case scenario settings are applied in the given order
scenario_config	The path where the scenario config table is stored.

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 *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")
```

Arguments

path	path of the main folder of the model
modulepath	path to the module folder 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 severals files into a single GAMS file.

Usage

```
singleGAMSfile(modelpath = ".", mainfile = "main.gms", output = "full.gms")
```

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.

Author(s)

Jan Philipp Dietrich, Anastasis Giannousakis

Examples

```
# copy dummymodel create single gms file out of it
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 *Creative 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

```
# copy dummymodel to temporary directory and compress it
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"
)
```

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

update_modules_embedding

Update Modules Embedding in GAMS code

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

```
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)
```

Index