Package 'vartors'

August 11, 2014

Type Package

Title Transform Definition of Variables to R Scripts

Version 0.2.6

Date 2014-08-10

Maintainer Joris Muller <joris.muller@etu.unistra.fr>

Description Generates R scripts useful to perform repetitive tasks on variables of different classes from a simple database. First it helps to create a tabular file describing the variables. Then the package could process this tabulate file to create a script in a .R or .Rmd format. This script will include code blocks for each variable according to the variable description. It could help to import, adapt to R classes and perform descriptive analysis on each variable according to its type. This R script could be used as it is or it could be modified if necessary to perform additional analysis. The user could write his own R script template to produce a customized script.

Depends R (>= 3.0.2)

Imports methods

Suggests testthat,roxygen2,utils,knitr

License GPL-3

URL https://github.com/jomuller/vartors

BugReports https://github.com/jomuller/vartors/issues/new

LazyData true

Encoding UTF-8

NeedsCompilation no

BuildVignettes true

VignetteBuilder knitr

R topics documented:

`bad_database` *Bad database to test vartors*

Description

A small dataset randomly generated to simulate an hypothetical survey and test **vartors**.

Format

A `data.frame` with 100 rows and 10 variables

Details

Include multiple definitions for NA's and not meaningful variables. It's a typical example of database we have to process. This database is close to example_df but more realistic because includes more typing errors. It is used in the vignette tutorial. This dataset was generated with the function `simulate_dataframe` from the package `dfexplore`, wrote in a `csv` file, altered to add errors and imported in R with `read.csv`.

The columns are :

`subject` An integer. Unique number of the subject.

`initial` A factor. Initials of the subject. Recognised as a factor by `read.csv` instead of a character vector.

`birth` A factor. Birthdate. Recognized as a factor by `read.csv` instead of a date.

`sex` A factor with levels `male` `female`

`study_level` A factor with levels `primary` < `secondary` < `superior` but recognized as a simple factor instead of a oredered factor.

`heigh` A factor. Recognized as a factor by `read.csv` instead of a numeric because there are multiple definitions for NA

`weight` A factor. Recognized as a factor by `read.csv` instead of a numeric because there are multiple definitions for NA

`siblings` A factor. Recognized as a factor by `read.csv` instead of a an integer because there are multiple definitions for NA

`Q1` An integer. Question 1. Without further description, we can't guess what's the meaning of this variable.

`Q2` An integer. Question 2. Without further description, we can't guess what's the meaning of this variable.

See Also

`variables_description_bad_database` is an example of variable description table for this database.

Examples

```
# See the class of each variable
str(bad_database)

# Create a variable description table skeleton
descvar_baddb <- descvars_skeleton(bad_database)

# Edit the variable description table
## Not run:
variables_description_bad_database <- edit(descvar_baddb)

## End(Not run)

# Watch the variable description table after editing
variables_description_bad_database

# Use it to create a script to import bad_database
myscript <- create_script(variables_description_bad_database)
## Not run:
# Show the script
myscript

# Write the script in a file
write_file(myscript, "my_import_script.R")

## End(Not run)
```

`c,ScriptOutput-method` *Concatenate ScriptOutput objects together.*

Description

Concatenate ScriptOutput objects together.

Usage

```
## S4 method for signature 'ScriptOutput'
c(x, ..., recursive = FALSE)
```

Arguments

x	The first ScriptOutput object.
...	Others ScriptOutput object to be concatened.
recursive	logical. If recursive = TRUE, the function recursively descends through lists (and pairlists) combining all their elements into a vector.

Details

This is a c method specific for ScriptOutput objects. All ScriptOutput objects must use the same language (.R or .Rmd). The header and the footer of the first object will be used for all. It is used in **vartors** by create_script to create a script for more than one variable.

Author(s)

Joris Muller

See Also

The c function and the ScriptOutput class.

Examples

```
# Create a script output from a description table
myscript <- create_script(variables_description_bad_database)

# But you forget a variable.
# Create it as a VariableDef object
forgoten_var <- vardef(varlabel = "A forgotten variable", rname = "forget", type = "integer")
forgoten_var

# Create a script for it
forgoten_script <- create_script(forgoten_var)
forgoten_script

# Add it to the initial script
my_complete_script <- c(myscript, forgoten_script)

# Watch the result
## Not run:
my_complete_script

# Write the script in a file
write_file(my_complete_script, "my_import_script.R")

## End(Not run)
```

create_script *Create a script*

Description

Create a script according to the definition of the variables and a template .

Usage

```
create_script(var_desc, template, rawdata_name = "raw_data",
  cleandata_name = "clean_data", header = TRUE, footer = TRUE, ...)

## S4 method for signature 'VariableDef,ScriptTemplate'
create_script(var_desc, template,
  rawdata_name = "raw_data", cleandata_name = "clean_data",
  header = FALSE, footer = FALSE, columns_names = var_desc$rname)

## S4 method for signature 'VariableDef,character'
create_script(var_desc, template,
  rawdata_name = "raw_data", cleandata_name = "clean_data")

## S4 method for signature 'VariableDef,ANY'
create_script(var_desc, rawdata_name = "raw_data",
  cleandata_name = "clean_data", header = FALSE, footer = FALSE)

## S4 method for signature 'DatabaseDef,ANY'
create_script(var_desc, template,
  rawdata_name = "raw_data", cleandata_name = "clean_data", header = TRUE,
  footer = TRUE)

## S4 method for signature 'data.frame,ANY'
create_script(var_desc, template,
  rawdata_name = "raw_data", cleandata_name = "clean_data", header = TRUE,
  footer = TRUE)
```

Arguments

var_desc	An object which describes the variable. Could be a single VariableDef, a whole DatabaseDef or a simple data.frame. In this last case, import_vardef function will be called to transform it to a DatabaseDef.
template	Optional. An object which describes the template. Either a ScriptTemplate object or a path to the template file. In this last case, import_template function will be called to transform this filepath to a ScriptTemplate. If missing, the default template is used.
rawdata_name	Name used to replace rep_rawdata in the template
cleandata_name	Name used to replace rep_cleandata in the template
header	If TRUE produce the header bloc
footer	If TRUE produce the footer bloc
columns_names	rnames of the columns
...	others arguments for specifics methods

Details

create_script is the central function of the vartors package. It will collate the two input objects created by the user (a VariableDef object or DatabaseDef object and a ScriptTemplate object) and will produce the final product : the script skeleton represented by an ScriptOutput object.

Value

A ScriptOutput object. This object could be written in a file with the write_file function.

Methods (by class)

- var_desc = VariableDef,template = ScriptTemplate:
- var_desc = VariableDef,template = character:
- var_desc = VariableDef,template = ANY:
- var_desc = DatabaseDef,template = ANY:
- var_desc = data.frame,template = ANY:

Author(s)

Joris Muller

See Also

import_template, import_vardef and the general documentation of vartors.

Examples

```
# Import a data.frame containing the description of the variables
# Show the description of the variable
sample_descvar

# Create the script skeleton simply with create_script()
script_skeleton <- create_script(sample_descvar)
# watch the result
script_skeleton
```

DatabaseDef-class *Class DatabaseDef*

Description

The DatabaseDef class is used to store properly the definition of several variables. It is created by import_vardef.

Slots

DatabaseDef A list containing VariableDef objects.

Note

For the moment, this class is only a convenient way to store a list of `VariableDef-class` objects. This class will be extended in a future version of **vartors** to add the pathfile, a global description and others informations about the database.

Author(s)

Joris Muller

See Also

`DatabaseDef` objects are created by import_vardef function for the moment. It store `link[=VariableDef-class]{Vari` objects in a list. To create a single definition of variable, use `vardef`.

Examples

```
# Create a DatabaseDef from a definition of variable table
suppressWarnings(
a_DatabaseDef_object <- import_vardef(sample_descvar)
)

# Show the list of the definition of variable
a_DatabaseDef_object

# Check the class
class(a_DatabaseDef_object)
```

---

descvars_skeleton *Skeleton of a definition of variables table*

Description

Create a *definition of variables table* skeleton in a data.frame from a database. Basically, this function gets the header of the database, puts it in the column "originalname", gets the type and put them in "type", adds column "spreadsheet column letter" and all the others columns to have a *definition of variables table*.

Usage

```
descvars_skeleton(database, factor_detect = 6)
```

Arguments

`database`	A data.frame with the data imported for example with `read.csv` or `read.xslx`.
`factor_detect`	An integer. If the number of unique value in a variable is below this threshold, then it will be considered as a factor

Value

Return a `data.frame`. This `data.frame` could be used as a skeleton of descvar, for example exporting it in a file with `write.csv` or `write.xlsx`

Author(s)

Joris Muller

Examples

```
# Import a database
data(example_df)
head(example_df)

# Create a skeleton of DatabaseDef from this database
descvars_sk <- descvars_skeleton(example_df)
descvars_sk[,1:10]

# This skeleton could be written on the disk in csv
# to be completed later in a spreadsheet sofware
## Not run:
write.csv(descvars_sk, file="Variables_description.csv")

## End(Not run)
# or in Excel
## Not run:
libary(openxslx)
write.xlsx(descvars_sk, file="Variables_description.xlsx")

## End(Not run)
```

---

example_df *Sample data.frame to test vartors*

Description

A small data set randomly generated to simulate 100 observation on an hypothetical survey with 10 columns. Include NA's.

The variables are :

- `subject` A numeric. Unique number of the subject.
- `initial` A character. Initials of the subject.
- `birth` Birthdate
- `sex` A factor with levels `male` `female`
- `study_level` An ordered factor with levels `primary` < `secondary` < `superior`
- `heigh` A numeric
- `weight` A numeric
- `siblings` A numeric
- `Q1` A numeric : question 1
- `Q2` A numeric : question 2

Format

A data frame with 100 rows and 10 variables

See Also

bad_database, a version closer to a real-life database.

`excel_skeleton` *Write an Excel VariableDef skeleton*

Description

Write an Excel VariableDef skeleton

Usage

```
excel_skeleton(filepath = "variables_description.xlsx")
```

Arguments

`filepath`	path to the file to create

`export_template` *Export a built-in script template*

Description

Export to a file a built-in script template. This way it's possible to adapt it to user's needs. Basically, it's a wrapper for the `file.copy` function.

Usage

```
export_template(builtin = "template_en.R", to = "./template_en.R")
```

Arguments

`builtin`	name of the built-in template. See details.
`to`	path where the file have to be written

Details

Actually, built-in templates are : anothertemplate.R, simpletemplate.R, template_en.R, template_fr.R, template_fr.Rmd

Value

Return the path where the file template was written. If there is an error, return FALSE.

Author(s)

Joris Muller

See Also

import_template

Examples

```
# export the default built-in template
## Not run:
export_template("template_to_edit.R", "en.R")

## End(Not run)
```

`import_template` *Import a script template*

Description

Import a script template and transform it as an ScriptTemplate-class object.

Usage

```
import_template(path, builtin, language = "R", idiom = "en",
  encoding = "UTF-8")
```

Arguments

path	Path to the template file
builtin	The name of a built-in template. See details for available built-in template. Override language and idiom parameters if given.
language	The name of one language of the built-in template, could be en or fr. If a path or a built-in is provided, this argument is ignored.
idiom	The idiom of the built-in template, could be en or fr. If a path or a built-in is provided, this argument is ignored.
encoding	Encoding of the script template. Should be "ASCII", "latin-1" or "UTF-8" (default value)

Details

Actually, built-in templates are : anothertemplate.R, simpletemplate.R, template_en.R, template_fr.R, template_fr.Rmd

Value

Return a ScriptTemplate object.

Author(s)

Joris Muller

See Also

ScriptTemplate-class, script_template

Examples

```
# import the default built-in template
import_template()

# import a specific built-in template
import_template(builtin = "anothertemplate.R")
```

`import_vardef` *Import definition of variables*

Description

Import definition of several variables and create a `DatabaseDef` object.

Usage

```
import_vardef(vardf, col_replacement)

## S4 method for signature 'data.frame'
import_vardef(vardf, col_replacement)
```

Arguments

`vardf`	A `data.frame` that represents a definition of variables table.
`col_replacement`	Replacement for the columns

Details

The `col_replacement` parameter by default are : `c(rname = "rname", varlabel = "varlabel", desc`
It is possible to overwrite by passing `c(key = "value")` in the `colnames` parameter.

Value

Return a `DatabaseDef` object.

Methods (by class)

- `data.frame`:

See Also

To create a definition of variables table from a database, use `/link{descvars_skeleton}`.

Examples

```
# create a simple definition of variables table in a data.frame
testdf <- read.table(header = TRUE, stringsAsFactors=FALSE,
  text="
  rname  varlabel  description  type  flevel1  name1  flevel2  name2  flevel3  name3
  id        Identification  'Unique ID'  integer NA NA NA NA NA NA
  age       'Age of patient' NA integer NA NA NA NA NA NA
  city      'City'  'City where live actually' factor 1 Strasbourg 2 Paris 3 London
  weight    'Weight' 'Weight at the beginning of the study' numeric NA NA NA NA NA NA
  ")
# create the DatabaseDef object
import_vardef(testdf)

# When the headers are not standard, it's possible to pass a
# replacement dictionnary
names(testdf) <- c("variable", "etiquette", "description",
                   "type",  "code1",  "modalite1",  "code2",  "modalite2",
                   "code3",  "modalite3")
head(testdf)
import_vardef(testdf,
              col_replacement =  c("rname" = "variable",
                                   "varlabel"    = "etiquette",
                                   "flevel"    = "code",
                                   "flabel"     = "modalite")
)
```

`list_templates` *List available vartors templates in a folder*

Description

List the file in the specified directory and check if these files are **vartors** templates. To detect, a file as **vartors** templates, the file must have one of the supported extension ('.R' or '.Rmd' for the moment) and have the tag `<vartors template>` in the first lines.

Usage

```
list_templates(dirpath)
```

Arguments

`dirpath`	path to the directory. If missing, the directory of the **vartors** package with built-in templates.

Value

Return a character vector with the names of the files which are **vartors** templates.

Author(s)

Joris Muller

See Also

Script templates could be imported with `import_template`. This function use `is_vartors_template` to check if a file is a vartors template.

`is_vartors_template`

Examples

```
# Get the list of built-in template
list_templates()
```

`sample_descvar` *Sample definition of variables table*

Description

A dataset containing definition of various variable with some errors. It's generated from a .CSV

- rname Short name of the variable to be use in R
- varlabel Long name of the variable used in graphs and tables
- description Description of the variable
- type Type of variable
- unit Unit of the variable or date format
- level One level of a qualitative variable
- name One name of a level of a qualitative variable

Format

A .csv with 6 rows and 13 variables

See Also

The documentation about definition of variables tables. To import it, use `import_vardef`.

`ScriptOutput-class` *Class ScriptOutput*

Description

Class to store the script output

Slots

`output` body of the script

`language` Language of the script. Is "R", "Rmd" or "Rnw"

`header` Header of the script

`footer` Footer of the script

Author(s)

Joris Muller

See Also

The constructor method is `create_script`. `ScriptOutput` objects have also a write method to create easily a file

---

`ScriptTemplate` *Script template*

Description

Script templates are a powerful concept in the **vartors** package. They are script skeletons which will be used to produce usable script thanks to definition of variables.

Details

Script templates should be written in different languages known by R :

R The classical R language, using file extension `.R`

R markdown Used to mix markdown syntax with R code. Use file extension `.Rmd`. Process these files with **knitr**

R sweave Used to mix LaTeX syntax with R code. Process these files with **knitr** too. No test where done with this format for the moment, because R markdown does almost everything in an easier way.

To be valid, a script template must contain `<vartors template>` in a comment somewhere in his 5 first lines.

To understand how to read and write a script template, there are two main concepts : *blocs* and *remplacements words*

blocs: Blocs are code lines between an opener delimiter and a closer delimiter. Openers are lines starting with `#<` and closers with `#>`. These delimiters must have a name recognized by `import_template` (actually, should be header, footer, integer, numeric, factor, ordered, date or not_used). Only one name by delimiter is allowed. For example, to create a new bloc for factor type, just write :

```
#< factor
# factors must use hist to make nice plots
plot(rep_cleandata$rname)
#> factor
```

This will add theses lines to the bloc of lines for the factor type

remplacement words: These words will be replaced when `create_script` will be used. They have a prefix `rep_`. For example, `rep_rname` will be replaced by the name of the variable in R from the definition of variables. Actually, usable replacement names are rep_rname, rep_type, rep_description and others ones...

See Also

The main methods are `import_template` and `export_template`. They are always used in `create_script`. To be usable in **vartors**, a script template must be transformed in a `ScriptTemplate` object by `create_script`.

`ScriptTemplate-class` *Class ScriptTemplate*

Description

The `ScriptTemplate` class is used to store properly the script template. It consists of various blocs, each for each type (numeric, factor...)

Slots

`language` A length-one character vector. The extension of the language used in the template. Should be `R`, `Rmd` or `Rnw`

`original_script` The original script

`blocs` A list. Each element of the list is a character vector with the lines for a type.

Author(s)

Joris Muller

See Also

The main function to construct a `ScriptTemplate` object is `import_template`. The constructor is `script_template`. More information about template in the dedicated documentation.

`vardef` *Create a VariableDef object*

Description

Constructor of the `VariableDef` class. A `VariableDef` object stores all the data needed to process a variable in the package.

Usage

```
vardef(varlabel, description, rname, type = "not_used", comment, unit,
  levels = NULL, names = NULL)
```

Arguments

varlabel	A character. Used to label properly the plots and tables in output.
description	A character. A description of the variable
rname	A character. It's the name of the variable used in R.
comment	A character.
type	A character. Must be one of the following : numeric, integer, factor, ordered, character, date or not_used.
unit	A character. Could be used for the format of a date (by default aa/mm/yyyy).
levels	A character vector. Describe the levels used for a vector.
names	A character vector of the same size than number of levels or empty.

Author(s)

Joris Muller

See Also

VariableDef-class and DatabaseDef-class

VariableDef-class *Class VariableDef*

Description

The VariableDef class is used to store properly the definition of the variable.

Slots

varlabel A length-one character vector. Should be with a max of 40 letters. All characters are allowed. Will be used to varlabel properly the plots and tables in output.

description A length-one character vector. Description of the variable.

rname A length-one character vector. Should be with a max of 16 letters. It's the name of the variable used in R. It could only use [a-z], [0-9] and "_" and must start with [a-z].

comment A length-one character vector with a max of 1000 letters. It's a commentary that will appear when describing each variable and give some advices to the statistician to how to analyze this variable.

type A length-one character vector. Must be one of the following : numeric, integer, factor, ordered, character, date or not_used. It's used to dispatch the script blocs regarding the type of the variable.

unit A length-one character vector of max size 20. Should be the unit of a variable which will be showed in some graphs or the format of a date (by default %d/%m/%Y).

levels A character vector. Only used if type is factor or ordered. Describe the levels used. The same levels must be in the database, otherwise NA will be generated.

labels A character vector of the same size than levels or empty. If empty, the labels will be the levels.

Author(s)

Joris Muller

See Also

The constructor is vardef. For several variables, see `DatabaseDef-class`

`variables_description_bad_database`
Sample definition of variables table

Description

A dataset containing definition of various variable linked to the database "bad_database". It's generated from a .CSV

- column The column name in the spreadsheet (A, B, C...)
- rname Short name of the variable to be use in R
- varlabel Long name of the variable used in graphs and tables
- description Description of the variable
- type Type of variable
- unit Unit of the variable or date format
- flevel One level of a qualitative variable. One for each level.
- fname One name of a level of a qualitative variable. One for each label.

Format

A data.frame with 6 rows and 18 variables

See Also

The documentation about definition of variables tables. To import it, use `import_vardef`.

`variable_definition_table`
Definition of variables table

Description

The definition of variables table is a way to describe each variable from a database in a table where each line represent a variable, and each column one of its characteristic. The idea is to be explicit about each variable by describing these characteristics :

varlabel An explicit name of the variable, but short enough to be displayed on figures and tables. Example : *Date of birth* or *Creatinine Clearance*

description An explicit description of the variable, if the varlabel is not explicit enough. It helps the statistician to understand the meaning of the variable. Example : *The Creatinine Clearance measured at the entry of the patient in the hospital*

comment An commentary to help the statistician. Example : *This quantitative variable can't have value superior to 20.*

unit The unit of the variable, when applicable. For dates, put the format like in R. Example : for the Creatinine Clearance, *ml/min*, for the Date of birth, *%d%m%Y*

flevel A level of a factor or ordered variable. Each level must be placed in a separated column. Then there are as much *flevel* as levels of the variable

flabel A label of a factor or ordered variable. Each level must be placed in a separated column. Then there are as much *flevel* as levels of the variable

type Class of the variable. Could be *numeric*, *integer*, *factor*, *ordered*, *date*, *character* or *not_used*

rname The name of the variable in R. If not given, the `varlabel` will be used and transformed to a compatible name

See Also

To create a *definition of variables table* skeleton from a existing data.frame, use the `descvars_skeleton` function. To read a *definition of variables table* from a data.frame to a DatabaseDef object, use the `import_vardef`. *Variable definition table* could be used directly as a `data.frame` by `create_script`.
A built-in example of a complete *definition of variables table* is the `variables_description_bad_database` that describes the `bad_database`.

`vartors` *Transform Definition of Variables to R Scripts*

Description

vartors is an R package that produces R script using definition of variables described by user. It could help to import, adapt to R classes and perform descriptive analysis on each variable according to its type.

Details

Documentation:

This page explain the main concepts in `vartors`. See also the vignettes. There is one with a tutorial :

```
vignette(topic = "usage", package = "vartors")
```

and one with the complete workflow

```
vignette(topic = "workflow", package = "vartors")
```

Motivation:

The package **vartors** was created to speed-up the error-prone and important cleaning data phase in context of the statistical consultations. These methodology consultations are an important part of our daily work. The idea is to help physicians of our hospital to process their data and make accurate analysis. In our workflow, the physician must come with a database (mainly an Excel or .csv file), a description of the variables and a good question. For the moment, we spend too much time to clean up data and not enough to analyze it. That's where **vartors** may help.

Workflow:

We will describe here in a compact way the workflow. For more details, see the documentation of each function and the vignettes.

The global workflow is :

1. Create a *definition of variables table*. The `descvars_skeleton` function could help you to initiate this. Fill all the characteristics of each variable, especially the *type*.
2. Import this *definition of variables table* in R if it was created in a spreadsheet program, for example with `read.csv`, `read.table` or `read.xlsx`, to have it in `data.frame`.
3. Use `create_script` to create a script according to the definition of each variable.
4. Write this script in a file with `write_file`
5. Adapt your new script to special cases, test it line by line, and produce a report, for example with **knitr**

It's possible to use various built-ins script templates in `.R` or `.Rmd` with the function `import_template`. The user could also create his own script templates by exporting a built-in one with `export_template`. It's a flexible way to allow each user to adapt and perform analysis on each variable as he want.

Note

For bugreports, features request, use the github issue tracking at `https://github.com/jomuller/vartors/issues`.

Special thanks to :

- The GMRC (Groupe Méthode en Recherche Clinique) team of Service de Santé Public, Hôpitaux Universitaires de Strasbourg :
  - Dr Erik-André Sauleau to supervise me during this work
  - Mickael Schaeffer for all these advices, bugtracking and coding-mate in another package
  - Pr Nicolas Meyer to accept me to work on this package and all his advices
  - Dr François Lefèbvre, Dr François Séverac, Maël Barthoulot, Pierre Collard Dutilleul for advices and bugtracking.
- Pr Bruno Falissard Master 2 courses and his contagious enthusiasm about R
- My classmates in the Master 2 *Méthodologie et Statistiques en Recherche Biomédicale* for nice debates and ideas.
- Christophe Genollini for his free manuals: Petit Manuel de S4, Construire un Package and R, Bonnes pratiques
- Hadley Wickham for his on-line book Advanced R and his helpful packages **roxygen2**, **devtools**, **testthat** and **ggplot2** used in this package.
- The R Core Team and particularly Uwe Ligges for his reviews and his tips.
- The peoples implicated in the free softwares and websites used to create this package : R, RStudio, Git, GitHub, StackOverflow.

Author(s)

Joris Muller

See Also

The mains methods are create_script, import_template and import_vardef. The main work to the user is to fill a *definition of variables table*.

Examples

```
# Import a data.frame containing the description of the variables
# Show the description of the variable
sample_descvar

# Create the script skeleton simply with create_script()
script_skeleton <- create_script(sample_descvar)
# watch the result
script_skeleton
# Could be written in a file with the write() method
## Not run:
write_file(script_skeleton)

## End(Not run)

# It's possible to create a simple script for a single variable
a_variable_definition <- vardef(varlabel = "Creatinine Clearance", rname ="creat", type="numeric" )
create_script(a_variable_definition)
```

---

write_file *Write file method*

Description

Write a file for the specified object

Usage

```
write_file(object, filepath, append = FALSE,
  encoding = getOption("encoding"), ...)

## S4 method for signature 'ScriptOutput'
write_file(object, filepath = "dmscript",
  append = FALSE, encoding = getOption("encoding"), ...)
```

Arguments

object	Object to be written as a ScriptOutput object.
filepath	Where to write the file. No need to add the extension, it will be put following the language of ScriptOutput. If an extension is given, then it will be used
append	Append Append the file. FALSE by default.
encoding	Character encoding to use. Use the default encoding if not specified.
...	other options for specific methods

Details

If the object is a ScriptOutput object, it will write a script skeleton file .

Value

Return invisibly the file path of the new file

Methods (by class)

- ScriptOutput:

Note

Will be extended to others **vartors** objects in future releases. I don't use the write function because it is not a S3 or a S4 method and it's hard to promote in a good way.

Author(s)

Joris Muller

See Also

ScriptOutput-class

Examples

```
# Import a data.frame containing the description of the variables
# Show the description of the variable
sample_descvar

# Create the script skeleton simply with create_script()
script_skeleton <- create_script(sample_descvar)
# watch the result
script_skeleton
# Could be written in a file with the write() method
## Not run:
write_file(script_skeleton)

## End(Not run)
```

Index