Package 'designr'

October 13, 2022

Type Package

Title Balanced Factorial Designs

Version 0.1.12

URL https://maxrabe.com/designr

BugReports https://github.com/mmrabe/designr/issues

Description
Generate balanced factorial designs with crossed and nested random and fixed effects <https://github.com/mmrabe/designr>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 3.5.0)

Imports MASS, tibble, crossdes, methods, dplyr, lme4,

Suggests knitr, rmarkdown, jsonlite, writexl, lmerTest, afex, ggplot2, gridExtra,

VignetteBuilder knitr

NeedsCompilation no

Author Maximilian M. Rabe [aut, cre] (<https://orcid.org/0000-0002-2556-5644>), Reinhold Kliegl [aut] (<https://orcid.org/0000-0002-0180-8488>), Daniel J. Schad [aut] (<https://orcid.org/0000-0003-2586-6823>)

Maintainer Maximilian M. Rabe <maximilian.rabe@uni-potsdam.de>

Repository CRAN

Date/Publication 2021-04-22 14:00:15 UTC

R topics documented:

`+,factorContainer,factorContainer-method`

Concatenate design factors and designs

Description

By adding factors and designs by +, a new design is created that contains all of the components.

Usage

```
## S4 method for signature 'factorContainer,factorContainer'
e1 + e2
```

Arguments

`e1, e2`	factor containers, such as factors or designs

Value

A factorDesign object

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))
```

`design.contrasts` *Retrieve contrast codes for a design*

Description

This function can be used to retrieve contrast codes based on experimental codes / planned observations.

Use this function to retrieve the names for contrasts in an experimental design, such as used by lm, lmer and many other regression models.

Usage

```
design.contrasts(
  design,
  factors = names(fixed.factors(design)),
  contrasts = NULL,
  expand = TRUE,
  rename_contrasts = "%1$s%2$s",
  intercept = FALSE,
  interactions = FALSE,
  include_random_levels = FALSE
)

contrast.names(design, ranfac = NULL, as_symbols = FALSE, ...)
```

Arguments

`design`	A design object
`factors`	Which fixed factors to include in the output
`contrasts`	Contrasts to use for each categorical factor. Should be a list named after the fixed effects and containing contrast matrices, such as the ones generated by the standard contrast functions. If NULL or the fixed factor is not found in the list, default contrasts are used (typically, treatment contrasts).
`expand`	If TRUE, a design matrix is returned. If FALSE, all factors (and interactions) with their respective levels are returned.
`rename_contrasts`	This is the pattern after which columns in the design matrix are named. By default, this is a direct concatenation of factor name and contrast name.
`intercept`	If TRUE, an intercept is added to the matrix. Its value is 1 for all observations.
`interactions`	If TRUE, interactions of fixed factors are included.
`include_random_levels`	If TRUE, levels of random factors are included in the matrix.
`ranfac`	Return random-effects contrast names for a given random factor. If NULL, return fixed-effects contrast names.
`as_symbols`	Return contrast names as symbols rather than strings (character vectors).
`...`	Arguments to pass on to design.contrasts()

Value

A design matrix (if expand==TRUE, default) or a list of factor levels (if expand==FALSE) for design.contrasts or contrast names for contrast.names.

Functions

- `contrast.names`: Retrieve contrast names for a design

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))


design.contrasts(des)

contrast.names(des)

stopifnot(contrast.names(des) == c("Factor11B", "Factor22B"))

contrast.names(des, as_symbols = TRUE)

design.contrasts(des, contrasts = list(Factor2 = contr.sum))

contrast.names(des, contrasts = list(Factor2 = contr.sum))
```

`designr` *designr*

Description

designr is an R package to create and simulate crossed factorial designs.

Details

The package supports factorial designs with an arbitrary number of fixed and random factors. Fixed factors are factors for which levels are known and typically defined by the experimenter, e.g. an experimental condition or a quasi-experimental variable such as a subject's age group. Conversely, the instances of random factors are usually not known before data collection. Examples for random factors are subjects or items in a typical psychological experiment, with the individual tested subjects and used items being the instances of those random factors.

See Also

`fixed.factor`, `random.factor`, `design.codes`

Examples

```
# A fixed-effects design without repeated measurement is created as easily as this:

design1 <-
  fixed.factor("Age", levels=c("young", "old")) +
  fixed.factor("Material",  levels=c("word", "image"))
design1

# As can be seen, this experimental design requires 4 observations.

# Adding random factors
# Assume we want to test different groups of subjects. Each subject will only be `old` or `young`
# but be tested with stimuli of both categories `word` and `image`. In a typical behavioral
# experiment, `Age` would now be a between-subject/within-item factor and `Material` a
# within-subject/between-item factor. In other words, `Material` is now nested within the
# instances of `Subject`, whereas `Subject` is grouped by `Age`.

design2 <-
  fixed.factor("Age", levels=c("young", "old")) +
  fixed.factor("Material",  levels=c("word", "image")) +
  random.factor("Subject", groups = "Age")
design.codes(design2)

# The minimal experimental design will still require 4 observations, assigning one subject to each
# level of the between-subject factor `Age`.
```

`factor.design` *Factorial Designs*

Description

The main function of this package is to create factorial designs with this function.

Usage

```
factor.design(...)
```

Arguments

`...`	Factors to add to the design.

Value

An instance of `factorDesign` with the complete factorial design and all fixed and random factors.

See Also

`random.factor` and `fixed.factor` for creating factors to add to the design. `output.design` and `write.design` for creating a useful summary and writing it into output files.

Examples

```
# To create an empty design:
design <- factor.design()

# To create a design for a recognition memory experiment in
# which each participant only sees either picture or words:
design <- factor.design(
    fixed.factor("type",levels=c("pic","word")),
    fixed.factor("status",levels=c("old","new")),
    random.factor("subject", groups="type"),
    random.factor("item", groups="type"),
    random.factor(c("subject","item"), groups="status")
)

# This is identical to:
design <- fixed.factor("type",levels=c("pic","word")) +
          fixed.factor("status",levels=c("old","new")) +
          random.factor("subject", groups="type") +
          random.factor("item", groups="type") +
          random.factor(c("subject","item"), groups="status")

# Or:
design <- factor.design(
   ~type(pic,word)+status(old,new)+subject[type]+item[type]+subject:item[status]
)

# You can also create a new design by adding more factors to an existing one:

design1 <- factor.design(~type(pic,word)+status(old,new)+subject[type]+item[type])
design2 <- design1 + random.factor(c("subject","item"), groups="status")
```

---

factorContainer-class *Design matrix S4 functions*

Description

Design matrix S4 functions

Usage

```
## S4 method for signature 'factorContainer'
show(object)
```

Arguments

object	A factorDesign object.

Methods (by generic)

- show: Display a factor container

Examples

```
des <- factor.design()
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))
```

---

factorDesign-class *S4 Methods for designFactor*

Description

S4 Methods for designFactor

Arguments

x	a factorDesign

Examples

```
x <- fixed.factor("Factor1", c("1A","1B"))
y <- random.factor("Subject", c("Factor1"))
```

---

fixed.factor *Fixed factors*

Description

This function creates an instance of fixedFactor to be used in a factorDesign. Fixed factors typically relate to (quasi-)experimental factors such as experimental conditions/manipulations, subject/item characteristics ect.

Usage

```
fixed.factor(
  name,
  levels,
  blocked = FALSE,
  character_as_factor = TRUE,
  is_ordered = FALSE,
  block_name = "%1$s.%2$d",
```

```
  groups = character(0),
  replications = 1L,
  assign = "latin.square",
  ...
)
```

Arguments

name	Name of the fixed factor.
levels	If not grouped, a vector of factor levels. Any atomic data type (character, logical, numeric, integer) can be used. If grouped, this should be a named list with each entry being a vector (as described before) and its name being a value of the grouping factor(s). If grouped within several factors, i.e. an interaction, the values constituting the names should be concatenated by colons (:), e.g. `list(`f1l1:f2l1`=1:2, `f1l2:f2l1`=3:4, ...)`. If for any group there are no levels specified, a warning will be issued and NA will be assigned as the value for this factor. If this is intended and the warning should be suppressed, please explicitly assign NA as the value for that group, e.g. `list(`f1l1:f2l1`=1:2, `f1l2:f2l1`=NA, ...)`.
blocked	Set this to TRUE if the levels of this factor are blocked. In that case, a factor is created whose factor levels are different sequences of the levels specified in the function call.
character_as_factor	If this is TRUE, character vectors passed in `levels` are automatically converted to a factor type.
is_ordered	Is this an ordered factor?
block_name	If `blocked = TRUE`, by default, there is not only a design matrix column created that contains the complete sequence of block levels but also a column for each position of the sequence with its assigned level. You may specify a different naming pattern using sprintf naming conventions. The first argument passed is the factor name and the second argument is the sequence position (starting at 1). The default column names will be `factor.1`, `factor.2`, etc. If NULL, no additional block columns are created.
groups	Names of fixed factors in which to nest this fixed factor (see *Nesting fixed factors*).
replications	Either a single integer or an integer vector of the same length as `levels` that is used to determine how many times each factor level should be repeated.
assign	If `blocked = TRUE`, you may specify a different method of rotating levels. The default if `'latin.square'` but `'permutations'`, `'williams'`, and `'random.order'` are also available.
...	more data to save as attributes

Value

An instance of `fixedFactor`.

Nesting Fixed Factors

If `groups` is used, the function will attempt to nest levels of the newly created factor within levels/interactions of the specified grouping factors. Note that nesting of fixed effects is only allowed within other fixed effects combinations but not within random effects. For each combination of the grouping factors, e.g. each group, you should specify an individual vector of levels (see above). If you fail to supply levels for any group, `NA`s will be assigned. This could result in unpredicted behavior when more factors are added. If you know what you are doing and would like to suppress the warning, please explicitly specify `NA` as the (only) value to assign to that group. At any rate, it is highly recommended to run sanity checks on the balancedness of the design if you are nesting fixed factors!

See Also

`random.factor`

Examples

```
fixed.factor("correct", levels=c(TRUE, FALSE))
fixed.factor("age", levels=c("child", "youth", "adult"))
fixed.factor("order", levels=c("task1", "task2", "task3"), blocked = TRUE, assign="latin.square")
```

`gibsonwu2013` *Gibson & Wu (2013)*

Description

The dataset `gibsonwu2013` contains data from self-paced reading in Chinese, comparing the processing of subject-extracted relative clauses (SRCs) and object-extracted relative clauses (ORCs) in supportive contexts.

References

Gibson, E., & Wu, H.-H. I. (2013). Processing Chinese relative clauses in context. *Language and Cognitive Processes, 28*, 125–155. doi: 10.1080/01690965.2010.536656

`is.randomFactor` *Checking factor design data types*

Description

Check if argument is a design factor (either random or fixed factor), specifically a random factor, a fixed factor or a factor design.

Usage

```
is.randomFactor(fac)

is.fixedFactor(fac)

is.factorDesign(fac)

is.designFactor(fac)
```

Arguments

fac	Object to check.

Value

TRUE or FALSE

Functions

- is.randomFactor: Check if argument is a random factor.
- is.fixedFactor: Check if argument is a fixed factor.
- is.factorDesign: Check if argument is a factor design.

Examples

```
x <- fixed.factor("factor", c("level1","level2"))
y <- random.factor("factor")

stopifnot(is.fixedFactor(x) && !is.randomFactor(x))
stopifnot(!is.fixedFactor(y) && is.randomFactor(y))
stopifnot(is.designFactor(x) && is.designFactor(y))
```

---

nobs,factorDesign-method

Retrieve the number of observations

Description

Retrieve the number of observations

Usage

```
## S4 method for signature 'factorDesign'
nobs(object)
```

Arguments

object	A designFactor object

Value

The number of observations

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))

nobs(des)

stopifnot(nobs(des) == 4)
```

---

output.design *Summary of Factor Designs*

Description

These functions return useful summaries of a factor design, including the design matrix itself as well as other parameters and a list of random factors as experimental units.

Usage

```
output.design(
  design,
  group_by = NULL,
  order_by = NULL,
  randomize = FALSE,
  rename_random = TRUE
)

design.formula(
  design,
  contrasts = NULL,
  expand_contrasts = !missing(contrasts),
  interactions = TRUE,
  intercepts = TRUE,
  response = "dv",
  env = parent.frame()
)

design.units(design, rename_random = TRUE, include_interactions = FALSE)
```

```
design.codes(
  design,
  group_by = NULL,
  order_by = names(random.factors(design, include_interactions = FALSE)),
  randomize = FALSE,
  rename_random = TRUE
)
```

Arguments

design	The factorDesign object to summarize.
group_by	If not NULL, the design matrix is grouped by these factors. Factors must be valid columns of the design matrix. If used, $codes will be a list matched to the entries in $groups.
order_by	If not NULL, output within each output group is ordered by these columns.
randomize	After ordering, remaining rows in the same order rank are randomly shuffled.
rename_random	Should random factor levels be renamed? If TRUE, levels are renamed as strings composed of the factor name and factor level (e.g., Subj01, Subj02, ...). FALSE disables renaming of random factor levels. Alternatively, you may provide a function which should accept the vectorized ID (integer) as a first argument and the name (single character value) of the random factor as second argument or ignore it. Functions such as as.double or as.integer *are* possible because they ignore the second argument and only convert the ID.
contrasts	The contrasts to override (NULL if none to override)
expand_contrasts	If TRUE, factors with more than one contrast are replaced by so many contrasts, i.e. the result contains the names of the individual contrasts, not of the factors.
interactions	Should fixed effects be additive or interactive?
intercepts	Should an intercept be included?
response	The left-hand side of the equation. Typically, this is just the response/dependent variable.
env	The environment in which to embed the formula
include_interactions	Whether to include random factor interactions (i.e., counterbalancing factors) in the output

Details

The function design.units returns the experimental units of the design. Those are defined by random factors and their levels. See units return value below.

design.codes returns a dataframe or tibble of all planned observations including each observation's experimental codes, i.e. fixed and random factor levels. If you group the output, a list is returned. See codes return value below.

design.formula returns a list of formulas suitable for regression analysis. Currently, formulas for lm and lme4 are returned. See formulas entry,

Value

`output.design` returns a list containing all output summaries, including the following named entities:

`codes` Either a `tibble` with all experimental codes or a list of `tibbles` of experimental codes. The list entries are matched to the rows of `$groups`.

`groups` If grouped, contains a tibble in which each row represents an output group, matched to the entries in $codes. If not grouped, this is `NULL`.

`ordered` If ordered, contains a vector of order criteria. If not ordered, this is `NULL`.

`randomized` Value of `randomized`.

`units` A list of random factors and their levels for this design as tibbles. Empty list if no random factors in the design.

`formulas` A list of possible model formulas for use with functions such as `lm()` and `lmer()`.

The functions `design.codes`, `design.formula` and `design.units` only return the values of the fields `codes` (a `tibble` or list or `tibbles` of experimental codes), `formulas` (a list of model formulas), and `units` (a list of random factors and their levels), respectively.

Functions

- `design.formula`: Retrieve only the model formulas suitable for the design
- `design.units`: Retrieve only the experimental units of a design
- `design.codes`: Retrieve only the codes of planned observations of an experimental design

See Also

`design.formula` for more options generating model formulae other than the suggested default ones in the summary.

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))

output.design(des)
design.codes(des)
design.units(des)
design.formula(des)
```

`random.factor` *Random factors*

Description

This function creates an instance of `randomFactor` to be used in a `factorDesign`. A random factor is typically related to an experimental unit such as Subject, Item, Experimenter, ect. and does not have preset levels.

Usage

```
random.factor(
  name,
  groups = character(0),
  instances = 1L,
  assign = "latin.square",
  ...
)
```

Arguments

`name`	Name of the random factor as a character vector. Typically, this should be a length-1 vector (i.e., a single string) but you may pass multiple names of random factors whose interaction is to be nested in groups (see *Assignment Constraints*).
`groups`	Names of fixed and random factors that are to be used as grouping (nesting/between) levels.
`instances`	Number of times (as a single integer value) each level (instantiation) is to be replicated.
`assign`	For random factor interactions, use this method for counterbalancing instance assignment (see Assignment Constraints)
`...`	Additional arguments to be stored as `extra` values.

Value

An instance of the class `randomFactor`.

Nesting Random Factors

A typical case of nesting in a psychological experiment is to vary a factor between subjects. That means that each subject would only be assigned to one condition of the nesting fixed factor (such as type of instruction). All other fixed factors that are not listed under ‘groups‘ are considered to vary within the random factor. Note that nesting increases the number of replications of the random factor.

Note that a random factor may be nested within fixed and/or other random factors but fixed factors may only be nested within levels of other fixed factors.

Assignment Constraints

A random interaction (a random factor instantiated with more than one name) governs the assignment of the co-occurence of the listed random factors. That means that, for example, `random.factor(c("Subject","Item"), groups="correct")` ensures that the assignment of a Subject and Item to one another occurs in only *one* of the conditions of `correct`. You may use `assign = '...'` to provide the method to counterbalance the assignment. By default, `'latin.square'` is used but you may also use `'random.order'` or `'permutations'`. Note that, depending on the assignment method you use, the constituting random factors (in this case `Subject` and `Item`) will be replicated n-times (`n` being the number of conditions of the nesting factors) for `'latin.square'` and `'random.order'` and n!-times for `'permutations'`.

See Also

`fixed.factor`

Examples

```
# A random factor Subject that nests factors blockOrder and gender,
# i.e. blockOrder and gender are "between-subject"

random.factor("Subject", groups=c("blockOrder", "gender"))

# A random factor Item without any grouping
random.factor("Item")
```

`random.factors` *Extract factors by type*

Description

From a given design, extract contained random or fixed factors as a list.

Usage

```
random.factors(design, include_interactions = TRUE)

fixed.factors(design)
```

Arguments

`design`	The factor design to check.
`include_interactions`	Should random factor interactions be included?

Value

A list of factors that are either fixed or random.

Functions

- random.factors: Return fixed factors

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))

random.factors(des)
fixed.factors(des)

stopifnot(setequal(names(random.factors(des)), c("Subject")))
stopifnot(setequal(names(fixed.factors(des)), c("Factor1","Factor2")))
```

---

show.factorContainer *Output a design factor summary*

Description

Output a design factor summary

Usage

```
show.factorContainer(object)
```

Arguments

object	the factor container to display

---

simLMM *Simulate data from a linear mixed-effects model*

Description

This function simulates artificial data from a linear mixed-effects model.

Usage

```
simLMM(
  formula,
  data = NULL,
  Fixef,
  VC_sd,
  CP = 0,
  LMM = NULL,
  empirical = FALSE,
  verbose = TRUE,
  family = "gaussian"
)
```

Arguments

formula	A formula as used in a call to the lmer function: a one-sided linear formula object describing both the fixed-effects and random-effects part of the model, with no response variable to the left of the ~ operator.
data	a data frame containing the variables named in formula.
Fixef	a vector of all fixed-effect model parameters.
VC_sd	standard deviations of the variance components for the random effects. This is a list of vectors, where different list entries reflect different grouping structures, and each vector contains standard deviations of variance components (random intercepts and random slopes) for one grouping factor. The last list entry is the standard deviation of the residual noise term (for gaussian or lognormal families only).
CP	correlation parameters of the random effects. If CP is a single number, then all CP are set to this same value. If CP is a vector of length equal to the number of grouping factor, then each vector entry specifies one fixed value for all CP associated with this grouping factor. Otherwise, CP can be a list of correlation matrices, which specifies a full correlation matrix for each grouping structure.
LMM	if a lmerMod object containing a fitted lmer model is provided, then simLMM uses the estimated model parameters for data simulation.
empirical	logical. If true, Fixef specify the empirical not population fixed effects parameters. empirical=TRUE does not work for residual Bernoulli noise, and not if continuous covariates are used.
verbose	logical. If TRUE (default), then information about the used model parameters is provided. If FALSE, then no output is generated.
family	string specifying the response distribution: "gaussian" (default) assumes a normal distribution, binomial specifies a Bernoulli distribution with a logit link function, "lognormal" specifies a log-normal distribution; with "lp", only the linear predictor is generated with no residual noise.

Examples

```
design <-
  fixed.factor("X", levels=c("X1", "X2")) +
  random.factor("Subj", instances=30)
dat <- design.codes(design)
contrasts(dat$X) <- c(-1, +1)

dat$ysim <- simLMM(formula = ~ 1 + X + (1 + X | Subj),
                   data = dat,
                   Fixef = c(200, 10),
                   VC_sd = list(c(30,10), 50),
                   CP = 0.3,
                   empirical = TRUE)

dat$Xn <- ifelse(dat$X=="X1",-1,1)
# lme4::lmer(ysim ~ Xn + (Xn || Subj), data=dat, control=lmerControl(calc.derivs=FALSE))
```

`subset,factorDesign-method`

Subset factor designs

Description

Subset factor designs

Usage

```
subset(x, ...)
```

Arguments

`x`	A factorDesign object
`...`	*subset*: Criteria along which to filter in planned observations / design matrix rows., *select*: Names of factors to keep in the design matrix

Value

Returns a `factorDesign` object with a subsetted design matrix

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))

subset(des, select = "Subject")
subset(des, Factor1 == "1A" | Factor2 == "2B", "Subject")
```

write.design *Write Design Files*

Description

This function writes a design into a set of files. For each random factor, a unit list is created that contains a list of all levels (instances) of the random factor and the factor levels to which that level is assigned. Moreover, code_files are created that contain a complete set of experimental codes.

Usage

```
write.design(
  design,
  group_by = NULL,
  order_by = NULL,
  randomize = FALSE,
 run_files = paste0("run", ifelse(length(group_by) > 0L, paste0("_", group_by, "-%",
    seq_along(group_by), "$s", collapse = ""), "")),
  code_files = "codes_%s",
  output_dir,
  output_handler,
  file_extension = NULL,
  ...
)

write.design.csv(..., quote = FALSE, row.names = FALSE)

write.design.xlsx(..., format_headers = FALSE)

write.design.json(..., dataframe = "columns")
```

Arguments

design	The factorDesign to be written into files.
group_by	Experimental codes are to be grouped by these factors. If NULL, all codes are written into one file. Also see output.design for grouping design output.
order_by	The experimental codes are to be ordered by these columns. Also see output.design for ordering design output.
randomize	After ordering, lines in the same order rank are to be shuffled randomly if set to TRUE.
run_files	The pattern to be used for the file names of the run_files (i.e., files containing the experimental codes). By default, file names are "run_Group1_Othergroup4.ext" ect.
code_files	Code files (files containing conditions for levels of random factors) are named after this pattern.

output_dir	All files are written into this directory.
output_handler	This is the function that is called to write the data frames. If using write.design.csv, this is utils::write.csv and if using write.design.json, this is jsonlite::write_json.
file_extension	This is the file_extension to be added after each file name. Use '' if no file_extension is to be added. If 'NULL', the file_extension is guessed from the output_handler used.
...	Other parameters to be passed on to write.design and the underlying output_handler.
quote, row.names	see utils::write.csv()
format_headers	see writexl::write_xlsx(), default is FALSE
dataframe	see jsonlite::write_json()

Functions

- write.design.csv: Using default settings for writing CSV files
- write.design.xlsx: Using default settings for writing XLSX files (using the writexl package)
- write.design.json: Using default settings for writing JSON files (using the jsonlite package)

See Also

output.design for use of order_by and group_by.

Examples

```
des <- fixed.factor("Factor1", c("1A","1B")) +
       fixed.factor("Factor2", c("2A","2B")) +
       random.factor("Subject", c("Factor1"))

# This writes a CSV file for each subject and a CSV list of subjects
write.design(des, group_by = "Subject", output_handler = write.csv, output_dir = tempdir())

# This writes a single CSV file for all subjects and a CSV list of subjects
write.design(des, output_handler = write.csv, output_dir = tempdir())
```

Index