Package 'psyverse'

March 26, 2020

Type Package

Title Decentralized Unequivocality in Psychological Science

Version 0.1.0

Maintainer Gjalt-Jorn Peters <gjalt-jorn@behaviorchange.eu>

License GPL (>= 3)

Description The constructs used to study the human psychology have many definitions and corresponding instructions for eliciting and coding qualitative data pertaining to constructs' content and for measuring the constructs. This plethora of definitions and instructions necessitates unequivocal reference to specific definitions and instructions in empirical and secondary research. This package implements a human- and machine-readable standard for specifying construct definitions and instructions for measurement and qualitative research based on 'YAML'. This standard facilitates systematic unequivocal reference to specific construct definitions and corresponding instructions in a decentralized manner (i.e. without requiring central curation; Peters (2020) <doi:10.31234/osf.io/xebhn>).

BugReports https://gitlab.com/r-packages/psyverse/issues

URL https://r-packages.gitlab.io/psyverse

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Depends R (>= 3.0.0)

Imports stats (>= 3.0.0), yaml (>= 2.1.19), yum (>= 0.0.1)

Suggests covr, DiagrammeR (>= 1.0.0), knitr, rmarkdown, testthat

VignetteBuilder knitr

NeedsCompilation no

Author Gjalt-Jorn Peters [aut, cre, ctb]

Repository CRAN

Date/Publication 2020-03-26 14:20:02 UTC

R topics documented:

apply_graph_theme 2
base30toNumeric 3
cat0 4
generate_construct_overview 4
generate_dct_template 6
generate_id 7
invert_id 7
load_dct_dir 8
parse_dct_specs 10
repeatStr 11
vecTxt 11

Index **13**

`apply_graph_theme` *Apply multiple DiagrammeR global graph attributes*

Description

Apply multiple DiagrammeR global graph attributes

Usage

```
apply_graph_theme(dctGraph, ...)
```

Arguments

`dctGraph`	The DiagrammeR::DiagrammeR graph to apply the attributes to.
`...`	One or more character vectors of length three, where the first element is the attribute, the second the value, and the third, the attribute type (`graph`, `node`, or `edge`).

Value

The DiagrammeR::DiagrammeR graph.

Examples

```
exampleSpec <-
  system.file("inst",
              "extdata",
              "example_dct_spec_1.dct",
              package="psyverse");
parsedSpecs <- load_dct_specs(exampleSpec);
dctGraph <- parsedSpecs$output$basic_graph;
dctGraph <- apply_graph_theme(dctGraph,
                              c("color", "#0000AA", "node"),
                              c("fillcolor", "#00FFFF", "node"));
```

base30toNumeric *Conversion between base10 and base30 & base36*

Description

The conversion functions from base10 to base30 are used by the generate_id() functions; the base36 functions are just left here for convenience.

Usage

```
base30toNumeric(x)

base36toNumeric(x)

numericToBase30(x)

numericToBase36(x)
```

Arguments

x	The vector to convert (numeric for the numericTo functions, character for the base30to and base36to funtions).

Details

The symbols to represent the 'base 30' system are the 0-9 followed by the alphabet without vowels but including the y. This vector is available as base30.

Value

The converted vector (numeric for the base30to and base36to funtions, character for the numericTo functions).

Examples

```
numericToBase30(654321);
base30toNumeric(numericToBase30(654321));
```

cat0 *Concatenate to screen without spaces*

Description

The cat0 function is to cat what paste0 is to paste; it simply makes concatenating many strings without a separator easier.

Usage

```
cat0(..., sep = "")
```

Arguments

...	The character vector(s) to print; passed to cat.
sep	The separator to pass to cat, of course, "" by default.

Value

Nothing (invisible NULL, like cat).

Examples

```
cat0("The first variable is '", names(mtcars)[1], "'.");
```

generate_construct_overview *Generate construct overviews and instruction overviews*

Description

These functions use a DCT specification to generate a construct overview or an instruction overview.

Usage

```
generate_construct_overview(
  dctSpec,
  include = c("definition", "measure_dev", "measure_code", "manipulate_dev",
    "manipulate_code", "aspect_dev", "aspect_code", "rel"),
  hideByDefault = NULL,
  divClass = "btn btn-secondary",
  headingLevel = 3,
  hyperlink_ucids = "Markdown",
  urlPrefix = "#"
)
```

```
generate_definitions_overview(
  dctSpecDf,
  headingLevel = 3,
  hyperlink_ucids = "Markdown",
  urlPrefix = "#"
)

generate_instruction_overview(
  dctSpecDf,
  type,
  headingLevel = 3,
  hyperlink_ucids = "Markdown",
  urlPrefix = "#"
)
```

Arguments

dctSpec	The DCT specification, as resulting from a call to load_dct_specs() or load_dct_dir().
include	Which elements to include in the construct overview.
hideByDefault	Which elements to hide by default.
divClass	The class of the button to collapse/expand sections.
headingLevel	The level of the heading in the Markdown output that is produces.
hyperlink_ucids	The type of hyperlinks to generate; must be a valid string. Currently, if the value is"Markdown" or "HTML", hyperlinks in the corresponding formats are produced, and if it is "none" (or, actually any other string value), nothing is produced.
urlPrefix	The prefix to insert before the URL in the produced hyperlink. The default, "#", results in a link to an anchor (an HTML a element) on the current page.
dctSpecDf	The DCT specification dataframer, as produced by a call to load_dct_specs() or load_dct_dir(), and stored within the resulting object.
type	For instruction overviews, the type of instruction to generate can be specified: must be one of "measure_dev", "measure_code", "manipulate_dev", "manipulate_code", "aspect_dev", or "aspect_code".

Value

A character string with the overview.

Examples

```
### Add example
```

`generate_dct_template` *DCT templates*

Description

These functions can generate one or more empty DCT templates.

Usage

```
generate_dct_template(
  prefix = paste(sample(letters, 4), collapse = ""),
  output = NULL,
  overwrite = FALSE,
  createDirs = FALSE,
  addComments = TRUE,
  stopOnIllegalChars = FALSE
)

generate_dct_templates(
  x,
  outputDir = NULL,
  createDirs = FALSE,
  addComments = FALSE,
  stopOnIllegalChars = FALSE
)
```

Arguments

`prefix`, `x`	The prefix (`prefix`) or vector of prefixes (`x`) to use.
`output`, `outputDir`	The filename or directory to which to write the templates.
`overwrite`	Whether to overwrite any existing files.
`createDirs`	Whether to recursively create the directories if the path specified in `output` or `outputPath` does not yet exist.
`addComments`	Whether to add comments to the DCT specification as extra explanation.
`stopOnIllegalChars`	DCT identifier prefixes can only contain upper- and lowercase letters and underscores. This argument specifies whether to remove illegal characters with a warning, or whether to throw an error (and stop) if illegal characters are found,

Value

The DCT template(s), either invisibly (if `output` or `outputDir` is specified) or visibly.

generate_id *Generate unique identifier(s)*

Description

To allow unique reference to constructs, they require unique identifiers. These functions generate such identifiers by combining one or more identifier prefixes (usually a human-readable construct name such as 'attitude') with a unique identifier based on the second the identifier was generated. The identifier prefix may only contain lowercase and uppercase letters and underscores.

Usage

```
generate_id(
  prefix = paste(sample(letters, 4), collapse = ""),
  stopOnIllegalChars = FALSE
)

generate_ids(x, stopOnIllegalChars = FALSE)
```

Arguments

prefix	An identifier prefix.
stopOnIllegalChars	Whether to base::stop() or produce a base::warning() when encountering illegal characters (i.e. anything other than a letter or underscore).
x	A vector of identifier prefixes.

Value

a character vector containing the identifier(s).

Examples

```
generate_id('attitude');
```

invert_id *Invert identifier*

Description

Invert the identifier (generated by generate_id() for one or more constructs. This means that the identifier prefix is stripped and the last part is converted back from base 30 to base 10.

Usage

```
invert_id(x)
```

Arguments

x	The identifier(s) as a character vector.

Value

The identifier(s) as a numeric vector.

Examples

```
invert_id(generate_id('example'));
```

---

load_dct_dir *Load DCT specifications from a file or multiple files*

Description

These function load DCT specifications from the YAML fragments in one (`load_dct_specs`) or multiple files (`load_dct_dir`).

Usage

```
load_dct_dir(
  path,
  recursive = TRUE,
  extension = "rock|dct",
  regex,
  dctContainer = "dct",
  headingLevel = 2,
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)

load_dct_specs(
  text,
  file,
  delimiterRegEx = "^---$",
  dctContainer = "dct",
  headingLevel = 2,
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)

## S3 method for class 'dct_specs'
print(x, ...)
```

```
## S3 method for class 'dct_specs'
plot(x, ...)
```

Arguments

path	The path containing the files to read.
recursive	Whether to also process subdirectories (TRUE) or not (FALSE).
extension	The extension of the files to read; files with other extensions will be ignored. Multiple extensions can be separated by a pipe (|).
regex	Instead of specifing an extension, it's also possible to specify a regular expression; only files matching this regular expression are read. If specified, regex takes precedece over extension,
dctContainer	The container of the DCT specifications in the YAML fragments. Because only DCT specifications are read that are stored in this container, the files can contain YAML fragments with other data, too, without interfering with the parsing of the DCT specifications.
headingLevel	The level of the Markdown headings that are produced.
delimiterRegEx	The regular expression used to locate YAML fragments
ignoreOddDelimiters	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
encoding	The encoding to use when calling readLines(). Set to NULL to let readLines() guess.
silent	Whether to be silent (TRUE) or informative (FALSE).
text, file	As text or file, you can specify a file to read with encoding encoding, which will then be read using base::readLines(). If the argument is named text, whether it is the path to an existing file is checked first, and if it is, that file is read. If the argument is named file, and it does not point to an existing file, an error is produced (useful if calling from other functions). A text should be a character vector where every element is a line of the original source (like provided by base::readLines()); although if a character vector of one element *and* including at least one newline character (\\n) is provided as text, it is split at the newline characters using base::strsplit(). Basically, this behavior means that the first argument can be either a character vector or the path to a file; and if you're specifying a file and you want to be certain that an error is thrown if it doesn't exist, make sure to name it file.
x	The parsed parsed_dct object.
...	Any other arguments are passed to the print command.

Details

load_dct_dir simply identifies all files and then calls load_dct_specs for each of them. load_dct_specs loads the YAML fragments containing the DCT specifications using yum::load_yaml_fragments() and then parses the DCT specifications into a visual representation as a DiagrammeR::DiagrammeR graph and Markdown documents with the instructions for creating measurement instruments or manipulations, and for coding measurement instruments, manipulations, or aspects of a construct.

Value

An object with the DiagrammeR::DiagrammeR graph stored in `output$basic_graph`, a DiagrammeR::DiagrammeR graph with a summary of which specifications are provided for each construct in `output$completeness_graph` and the instructions in `output$instr`.

Examples

```
exampleSpec <-
  system.file("inst",
              "extdata",
              "example_dct_spec_1.dct",
              package="psyverse");
load_dct_specs(exampleSpec);

## Not run:
psyverse::load_dct_dir(path="A:/some/path");

## End(Not run)
```

`parse_dct_specs` *Parse DCT specifications*

Description

This function parses DCT specifications; it's normally called by `load_dct_dir()` or `load_dct_specs()`, so you won't have to use it directly.

Usage

```
parse_dct_specs(
  dctSpecs,
  headingLevel = 2,
  hyperlink_ucids = "Markdown",
  urlPrefix = "#"
)
```

Arguments

`dctSpecs`	The DCT specifications (a list).
`headingLevel`	The heading level for Markdown output.
`hyperlink_ucids, urlPrefix`	Passed on to the `generate_instruction_overview()` and `generate_construct_overview()` functions.

Value

The object of parsed DCT specifications.

repeatStr *Repeat a string a number of times*

Description

Repeat a string a number of times

Usage

```
repeatStr(n = 1, str = " ")
```

Arguments

n, str	Normally, respectively the frequency with which to repeat the string and the string to repeat; but the order of the inputs can be switched as well.

Value

A character vector of length 1.

Examples

```
### 10 spaces:
repStr(10);

### Three euro symbols:
repStr("\u20ac", 3);
```

vecTxt *Easily parse a vector into a character value*

Description

Easily parse a vector into a character value

Usage

```
vecTxt(
  vector,
  delimiter = ", ",
  useQuote = "",
  firstDelimiter = NULL,
  lastDelimiter = " & ",
  firstElements = 0,
  lastElements = 1,
  lastHasPrecedence = TRUE
```

```
)

vecTxtQ(vector, useQuote = "'", ...)
```

Arguments

`vector`	The vector to process.
`delimiter, firstDelimiter, lastDelimiter`	The delimiters to use for respectively the middle, first `firstElements`, and last `lastElements` elements.
`useQuote`	This character string is pre- and appended to all elements; so use this to quote all elements (`useQuote="'"`), doublequote all elements (`useQuote='"'`), or anything else (e.g. `useQuote='|'`). The only difference between `vecTxt` and `vecTxtQ` is that the latter by default quotes the elements.
`firstElements, lastElements`	The number of elements for which to use the first respective last delimiters
`lastHasPrecedence`	If the vector is very short, it's possible that the sum of firstElements and lastElements is larger than the vector length. In that case, downwardly adjust the number of elements to separate with the first delimiter (`TRUE`) or the number of elements to separate with the last delimiter (`FALSE`)?
`...`	Any addition arguments to `vecTxtQ` are passed on to `vecTxt`.

Value

A character vector of length 1.

Examples

```
vecTxtQ(names(mtcars));
```

Index

apply_graph_theme, 2

base30and36conversion
(base30toNumeric), 3
base30toNumeric, 3
base36toNumeric (base30toNumeric), 3
base::readLines(), 9
base::stop(), 7
base::strsplit(), 9
base::warning(), 7

cat, 4
cat0, 4

DiagrammeR::DiagrammeR, 2, 9, 10

generate_construct_overview, 4
generate_construct_overview(), 10
generate_dct_template, 6
generate_dct_templates
(generate_dct_template), 6
generate_definitions_overview
(generate_construct_overview),
4
generate_id, 7
generate_id(), 3, 7
generate_ids (generate_id), 7
generate_instruction_overview
(generate_construct_overview),
4
generate_instruction_overview(), 10

invert_id, 7

load_dct_dir, 8
load_dct_dir(), 5, 10
load_dct_specs (load_dct_dir), 8
load_dct_specs(), 5, 10

numericToBase30 (base30toNumeric), 3
numericToBase36 (base30toNumeric), 3

parse_dct_specs, 10
plot.dct_specs (load_dct_dir), 8
print.dct_specs (load_dct_dir), 8

readLines(), 9
repeatStr, 11
repStr (repeatStr), 11

vecTxt, 11
vecTxtQ (vecTxt), 11

yum::load_yaml_fragments(), 9