Package ‘justifier’

July 12, 2021

Title Human and Machine-Readable Justifications and Justified Decisions Based on ‘YAML’

Version 0.2.0

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

Description Leverages the 'yum' package to implement a 'YAML' ('YAML Ain't Markup Language', a human friendly standard for data serialization; see <https://yaml.org>) standard for documenting justifications, such as for decisions taken during the planning, execution and analysis of a study or during the development of a behavior change intervention as illustrated by Marques & Peters (2019) <doi:10.17605/osf.io/ndxha>. These justifications are both human- and machine-readable, facilitating efficient extraction and organisation.

License GPL (>= 2)

Encoding UTF-8

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

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

Imports data.tree (>= 0.7.8), DiagrammeR (>= 1.0.0), DiagrammeRsvg (>= 0.1), purrr (>= 0.3.0), yaml (>= 2.2.0), yum (>= 0.0.1)

Suggests covr, here, knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Gjalt-Jorn Ygram Peters [aut, cre]
  (<https://orcid.org/0000-0002-0336-9589>),
  Szilvia Zorgo [ctb] (<https://orcid.org/0000-0002-6916-2097>)

Repository CRAN

Date/Publication 2021-07-12 08:00:02 UTC
apply_graph_theme

Description

Apply multiple DiagrammeR global graph attributes

Usage

apply_graph_theme(graph, ...)

Arguments

graph 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

```r
exampleJustifier <- '---
assertion:
  - id: assertion_id
    label: "An assertion"
decision:
  - id: decision_id
    label: "A decision"
justification:
  - id: justification_id
    label: "A justification"
assertion:
  - id: assertion_id
    description: "A description of an assertion"
source:
  - id: source1_id
    label: "First source"
  - id: source2_id
    label: "second source"
---
justifications <- justifier::load_justifications(text=exampleJustifier);
miniGraph_original <- justifications$decisionGraphs[[1]];
miniGraph <- justifier::apply_graph_theme(
  miniGraph_original,
  c("color", "#0000AA", "node"),
  c("shape", "triangle", "node"),
  c("fontcolor", "#FF0000", "node")
);
### This line shouldn't be run when executing this example as test,
### because rendering a DiagrammeR graph takes quite long
## Not run:
DiagrammeR::render_graph(miniGraph);
## End(Not run)
```
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)
numericToBase30(x)
```

Arguments
- **x**: The vector to convert (numeric for the `numericTo` functions, character for the `base30to` and `base36to` functions).

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` functions, 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

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

Description

Clean your workspace

Usage

```r
clean_workspace(force = FALSE, silent = justifier::opts$get("silent"))
```

Arguments

- `force`: Whether to force cleaning the workspace
- `silent`: Whether to be chatty or silent.

Examples

```r
### Without `force=TRUE`, presents a query to the user in
### interactive mode:
clean_workspace(silent=FALSE);

### Set `force=TRUE` to force clean the workspace
clean_workspace(force = TRUE, silent=FALSE);
```

Description

Export justification as YAML

Usage

```r
export_justification(  
  x,  
  file = NULL,  
  encoding = "UTF-8",  
  append = TRUE,  
  preventOverwriting = TRUE,  
  silent = justifier::opts$get("silent")
)
```
Arguments

- **x**: The justification, either loaded from one or more files or programmatically constructed. This can be one or more decisions, justifications, assertions, or sources.
- **file**: If specified, the file to export the justification to.
- **encoding**: The encoding to use when writing the file.
- **append**: Whether to append to the file, or replace its contents.
- **preventOverwriting**: Whether to prevent overwriting an existing file.
- **silent**: Whether to be silent or chatty.

Value

The generated YAML, invisibly, unless file is NULL.

Examples

```r
### Programmatically create a simple justification object
justifierObject <-
  justifier::asrt(
    "assertion",
    source = c(
      justifier::srce('/quotesingle.Var source1'),
      justifier::srce('/quotesingle.Var source2')));

### Export to YAML
justifierYAML <-
  justifier::export_justification(
    justifierObject,
    file=NULL);

### Show YAML
cat(justifierYAML, sep="\n");
```

---

**generate_id**  
*Generate unique identifier(s)*

Description

Convenience function to generate a unique identifiers for sources, assertions, justifications, and decisions.

Usage

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

type
 prefix
 stopOnIllegalChars

Arguments
 type
 prefix
 stopOnIllegalChars

Value

A character vector containing the identifier(s).

Examples

generate_id(type = "S", 'sourceExample');
generate_id(type = "A", 'assertionExample');

get_workspace

Get your justifier workspace identifier

Description

This is used to be able to log decisions programmatically.

Usage

get_workspace(silent = justifier::opts$get("silent"))

Arguments

silent

Value

Invisibly, the workspace identifier.

Examples

justifier::get_workspace_id();


### get_workspace_id

*Get your justifier workspace identifier*

**Description**

This is used to be able to log decisions programmatically.

**Usage**

```r
get_workspace_id(silent = justifier::opts$get("silent"))
```

**Arguments**

- `silent` Whether to suppress messages.

**Value**

Invisibly, the workspace identifier.

**Examples**

```r
justifier::get_workspace_id();
```

### ifelseObj

*Conditional returning of an object*

**Description**

The ifelseObj function just evaluates a condition, returning one object if it's true, and another if it's false.

**Usage**

```r
ifelseObj(condition, ifTrue, ifFalse)
```

**Arguments**

- `condition` Condition to evaluate.
- `ifTrue` Object to return if the condition is true.
- `ifFalse` Object to return if the condition is false.

**Value**

One of the two objects
load_justifications

Examples

dat <- ifelseObj(sample(c(TRUE, FALSE), 1), mtcars, Orange);

load_justifications

Load Justifications from a file or multiple files

Description

These functions load justifications from the YAML fragments in one (load_justifications) or multiple files (load_justifications_dir).

Usage

load_justifications(
  text = NULL,
  file = NULL,
  delimiterRegEx = "^---$",
  justificationContainer = c("justifier", "justification", "decision", "assertion", "source"),
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  storeDecisionGraphSvg = TRUE,
  silent = TRUE
)

load_justifications_dir(
  path,
  recursive = TRUE,
  extension = "jmd",
  regex = NULL,
  justificationContainer = c("justifier", "justification", "decision", "assertion", "source"),
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)

Arguments

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.

delimiterRegEx  The regular expression used to locate YAML fragments
justificationContainer
  The container of the justifications in the YAML fragments. Because only justi-
fications 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 justi-
fications.
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.
storeDecisionGraphSvg  Whether to also produce (and return) the SVG for the decision graph.
silent  Whether to be silent (TRUE) or informative (FALSE).
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 specifying an extension, it’s also possible to specify a regular expres-
sion; only files matching this regular expression are read. If specified, `regex`
takes precedence over `extension`.

Details

`load_justifications_dir` simply identifies all files and then calls `load_justifications`
for each of them. `load_justifications` loads the YAML fragments containing the justifications using
`yum::load_yaml_fragments()` and then parses the justifications into a visual representation as a
`ggplot2::ggplot` graph and Markdown documents with overviews.

Value

An object with the `ggplot2::ggplot` graph stored in `output$graph` and the overview in `output$overview`.

Examples

```r
exampleMinutes <- 'This is an example of minutes that include
a source, an assertion, and a justification. For example, in
the meeting, we can discuss the assertion that sleep deprivation
affects decision making. We could quickly enter this assertion in
a machine-readable way in this manner:
```
Because it is important to refer to sources, we cite a source as well. We have maybe specified that source elsewhere, for example in the minutes of our last meeting. That specification may have looked like this:

```r
---
source:
  - id: source_Harrison
    xdoi: "doi:10.1037/1076-898x.6.3.236"
    type: "Journal article"
---
```

We can now refer to these two specifications later on, for example to justify decisions we take.

```r
justifier::load_justifications(text=exampleMinutes);
```

### To load a directory with justifications

```r
eexamplePath <- file.path(system.file(package="justifier"), 'extdata');
jjustifier::load_justifications_dir(path=examplePath);
```

---

### Description

Used to programmatically document decisions - note that you have to store them to a file to not lose them (i.e. if used interactively).

### Usage

```r
log_decision(
  label,
  description = "",
)```
arguments = "", 
    date = as.character(Sys.Date()), 
    id = NULL, 
    justification = "", 
    silent = justifier::opts$get("silent"), 
...  
)

Arguments

label A human-readable label for the decision,
description A human-readable description.
alternatives The alternatives between which was chosen.
date The date of the decision.
id Optionally, a manually specified id (otherwise, randomly generated).
justification A justification specified using jstf(), or more than one, combined with the c operator.
silent Whether to print messages.
... Any additional options will be stored in the decision.

Value

Invisibly, the decision as a justifier object (generated by dcsn()).

Examples

clean_workspace(force = TRUE, silent=FALSE);
log_decision("First we start using `justifier`.", 
    silent=FALSE);
log_decision(paste0("Then we start documenting our ", 
    "decisions and justifications."), 
    silent=FALSE);
log_decision("Then we start learning from ourselves.", 
    silent=FALSE);
workspace();

merge_specLists Merging to justifier specification lists

Description

Merging to justifier specification lists

Usage

merge_specLists(x, y)
opts

Arguments

\( x, y \quad \text{The two justifier specification lists} \)

Value

A merged justifier specification list.

Examples

### Add example

---

<table>
<thead>
<tr>
<th>opts</th>
<th>Options for the justifier package</th>
</tr>
</thead>
</table>

Description

The `justifier::opts` object contains three functions to set, get, and reset options used by the `escalc` package. Use `justifier::opts$set` to set options, `justifier::opts$get` to get options, or `justifier::opts$reset` to reset specific or all options to their default values.

Usage

```r
opts
```

Format

An object of class `list` of length 4.

Details

If you use `justifier` to programmatically document your decisions in an R file, there is one option that you commonly use: `workspace_id` and `workspace_option_name`.

It is normally not necessary to get or set `justifier` options.

The following arguments can be passed:

... For `justifier::opts$set`, the dots can be used to specify the options to set, in the format `option = value`, for example, `EFFECTSIZE_POINTESTIMATE_NAME_IN_DF = "\n"`. For `justifier::opts$reset`, a list of options to be reset can be passed.

**option** For `justifier::opts$set`, the name of the option to set.

**default** For `justifier::opts$get`, the default value to return if the option has not been manually specified.

The following options can be set:

The name of the column with the effect size values.
The name of the column with the effect size variance.
The name of the column with the missing values.
### Get the default 'silent' setting
justifier::opts$get('silent');

### Set to FALSE
justifier::opts)set(silent = FALSE);

### Check that it worked
justifier::opts$get('silent');

### Reset this option to its default value
justifier::opts/reset('silent');

### Check that the reset worked, too
justifier::opts$get('silent');

---

**parse_justifications**  Parsing justifications

---

**Description**

This function is normally called by `load_justifications()`; however, sometimes it may be desirable to parse justifications embedded in more complex objects, for example as provided by `yum::load_and_simplify()`. Therefore, this function can also be called directly.

**Usage**

```r
parse_justifications(
  x,
  justifierFields = "^date$|^framework$",
  fromFile = NULL,
  path = NULL,
  storeDecisionGraphSvg = FALSE,
  silent = TRUE
)
```

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

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

**Arguments**

- `x` An object resulting from a call to `yum::load_and_simplify()`.
justifierFields
  Which fields to copy from justifier metadata to the elements within the specified scope.
fromFile  
The file from which the justifier specifications were read.
path     
The path holding these justifier specifications (not necessary if fromFile is provided).
storeDecisionGraphSvg
  Whether to also produce (and return) the SVG for the decision graph.
silent   
  Whether to be chatty or quiet.
...
  Additional arguments are passed on to graphics::plot() for the print method or to DiagrammeR::render_graph() for the plot method.

Details

While there is some flexibility in how justifications can be specified, they are most easily processed further if they all follow the same conventions. This function ensures this. The convention is as follows:

- all specifications are provided in four 'flat' lists, named after the types of elements they contain;
- all elements have a unique identifier
- all references to other elements are indeed only references to the other elements’ id’s in these 'flat lists'

Value

The parsed justifier object.

Examples

### Specify an example text
exampleFile <-
  system.file("extdata",
    "simple-example.jmd",
    package="justifier");

### Show contents
cat(readLines(exampleFile), sep="\n");

### Load it with yum::load_and_simplify()
loadedMinutes <- yum::load_and_simplify(exampleFile);

### Show contents
names(loadedMinutes);

### Parse 'manually'
parsedJustifications <- justifier::parse_justifications(loadedMinutes);

### Show contents
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("€", 3);

sanitize_for_DiagrammeR

Sanitize for DiagrammeR

Description

Basically a wrapper for \texttt{gsub()} to sanitize a string for DiagrammeR

Usage

sanitize_for_DiagrammeR(
  x,
  regExReplacements = justifier::opts$get("regExReplacements")
)
Arguments

x The string or vector
regExReplacements A list of two-element character vectors; first element should be the element to search, and the second element, the replacement.

Value
The sanitized character vector

Examples

justifier::sanitize_for_DiagrammeR("This is or isn't problematic");

Description
Save your workspace

Usage

save_workspace(
    file = NULL,
    encoding = "UTF-8",
    append = FALSE,
    preventOverwriting = TRUE,
    silent = justifier::opts$get("silent")
)

Arguments

file If specified, the file to export the justification to.
encoding The encoding to use when writing the file.
append Whether to append to the file, or replace its contents.
preventOverwriting Whether to prevent overwriting an existing file.
silent Whether to be silent or chatty.

Value
The result of a call to export_justification().
Examples

clean_workspace(force = TRUE, silent=FALSE);
log_decision("First we start using \`justifier`.",
    silent=FALSE);
log_decision(paste0("Then we start documenting our ",
    "decisions and justifications."),
    silent=FALSE);
log_decision("Then we start learning from ourselves.",
    silent=FALSE);
save_workspace();

set_workspace_id

Description

This is used to be able to log decisions programmatically.

Usage

set_workspace_id(id, silent = justifier::opts$get("silent"))

Arguments

id The workspace identifier
silent Whether to be suppress messages.

Value

Invisibly, the passed id.

Examples

set_workspace_id("my_workspace");

source

Programmatically constructing justifier elements

Description

These functions can be used to programmatically construct justifications.
source

Usage

source(label, description = NULL, type = NULL, id = NULL, xdoi = NULL, ...)

assert(label, description = "", type = NULL, id = NULL, source = NULL, ...)

justify(label, description = "", type = NULL, id = NULL, assertion = NULL, ...)

decide(
  label,
  description = NULL,
  type = NULL,
  id = NULL,
  alternatives = NULL,
  justification = NULL,
  ...
)

## S3 method for class 'justifierElement'
c(…)

Arguments

label  A human-readable label for the decision, justification, assertion, or source. Labels are brief summaries of the core of the decision, justification, assertion, or source. More details, background information, context, and other comments can be placed in the description.

description  A human-readable description. This can be used to elaborate on the label. Note that the label should be reader-friendly and self-contained; but because they also have to be as short as possible, descriptions can be used to provide definitions, context, background information, or add any other metadata or comments.

type  Types are used when working with a framework. Frameworks define type identifiers, consisting of letters, digits, and underscores. By specifying these identifiers the type of a decision, justification, assertion, or source. Source types can be, for example, types of documents or other data providers, such as "empirical evidence", "expert consensus", "personal opinion", or "that one meeting that we had in May". Assertion types can be, for example, data types or types of facts, such as 'number', 'prevalence', 'causal relationship', or 'contact information'. Justification types can be, for example, types of reasoning or logical expressions, such as 'deduction', 'induction', or 'intersection'. Decision types are the most framework-specific, heavily depend on the specific context of the decision, and are used by frameworks to organise the decisions in a project. Examples of decision types are the decision to recruit a certain number of participants in a scientific study; the decision to target a certain belief in a behavior change intervention; the decision to merge two codes in a qualitative study; the decision to hire a staff member; or the decision to make a certain purchase.

id  The identifier (randomly generated if omitted).
For sources, XDOI identifier (a DOI, or, if that does not exist, ISBN or other unique identifier of the source).

Additional fields and values to store in the element.

In assertions, the source (or sources) that the assertion is based on can be specified using src().

In justifications, the assertion (or assertions) that the justification is based on can be specified using asrt().

The alternatives that were considered in a decision.

In decisions, the justification (or justifications) that the decision is based on can be specified using jstf().

The generated object.

### Programmatically create a partial justification object
```r
justifierAssertion <-
  justifier::assert(
    "This is an assertion",
    source = c(
      justifier::source("This is a first source"),
      justifier::source("This is a second source")));
```

### Programmatically create a justification with two assertions
### but without sources
```r
justifierJustification <-
  justifier::justify(
    "Icecream will make me feel less fit",
    assertion = c(
      justifier::assert("Icecream is rich in energy"),
      justifier::assert("Consuming high-energy foods makes me feel less fit"),
    ),
    weight = -.5
  );
```

### Show it
```r
justifierJustification;
```

### Programmatically create a simple decision
```r
simpleDecision <-
  justifier::decide(
    "decision",
    justification = justifier::jstf(
      "justification",
      assertion = justifierAssertion
    )
  );
```
### Programmatically create a justification object for a full decision

```r
fullJustifierObject <-
  justifier::decide(
    "I decide to go get an icecream",
    justification = c(
      justifier::justify(
        "Having an icecream now would make me happy",
        assertion = c(
          justifier::assert(
            "Decreasing hunger increases happiness",
            source = justifier::source("My past experiences")
          ),
          justifier::assert(
            "I feel hungry",
            source = justifier::source("Bodily sensations")
          )
        ),
        weight = 1
      ),
      justifierJustification,
      justifier::justify(
        "I can afford to buy an icecream.",
        assertion = c(
          justifier::assert(
            "My bank account balance is over 300 euro.",
            source = justifier::source("My bank app")
          ),
          justifier::assert(
            "I need to keep at least 100 euro in my bank account.",
            source = justifier::source("Parental advice")
          )
        ),
        weight = .3
      )
    )
  )

### Show the full object

fullJustifierObject;
```

### Combine both into a list of decisions

```r
twoDecisions <-
c(simpleDecision,
  fullJustifierObject);
```
### Show the combination
twoDecisions;

---

#### to_specList

Producing a list of specifications

### Description

This function is for internal use, but has been exported in case it’s useful for people working ‘manually’ with lists of justifications.

### Usage

```r
to_specList(x, types, type, idsRequired = TRUE, silent = TRUE)
```

### Arguments

- `x`: The list to parse.
- `types`: The class to assign to the specification list (the `justifierSpecList` object to return).
- `type`: The class to assign to each specification (in addition to `justifierSpec`).
- `idsRequired`: Whether to require identifiers.
- `silent`: Whether to be chatty or silent.

### Value

A list of classes `c("justifierSpecList", types)` where each element is a specification of class `c("justifierSpec", type)`.

### Examples

```r
### Specify an example text
exampleFile <-
  system.file("extdata",
    "simple-example.jmd",
    package="justifier");

### Show contents
cat(readLines(exampleFile), sep="\n");

### Load it with yum::load_and_simplify()
loadedMinutes <- yum::load_and_simplify(exampleFile);

### Show contents
names(loadedMinutes);

### Show classes
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

### Convert to specification list
res <- to_specList(loadedMinutes["assertion"],
  type="assertion",
  types="assertions");

### Show classes
class(res);

### Show original and parsed objects
loadedMinutes["assertion"]; res;

Easily parse a vector into a character value
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));

workspace
Show your workspace contents

Description
Show your workspace contents

Usage
workspace(silent = justifier::opts$get("silent"))

Arguments
silent Whether to be chatty or silent.

Value
The workspace contents.

Examples
justifier::clean_workspace(force = TRUE, silent=FALSE);
justifier::log_decision("First we start using ‘justifier’.",
    silent=FALSE);
justifier::log_decision(paste0("Then we start documenting our ",
    "decisions and justifications."),
    silent=FALSE);
justifier::log_decision("Then we start learning from ourselves.",
    silent=FALSE);
justifier::workspace();
wrapVector | Wrap all elements in a vector

**Description**

Wrap all elements in a vector

**Usage**

```r
wrapVector(x, width = 0.9 * getOption("width"), sep = "\n", ...)
```

**Arguments**

- `x`: The character vector
- `width`: The number of
- `sep`: The glue with which to combine the new lines
- `...`: Other arguments are passed to `strwrap()`.

**Value**

A character vector

**Examples**

```r
res <- wrapVector(
  c(
    "This is a sentence ready for wrapping",
    "So is this one, although it's a bit longer"
  ),
  width = 10
);

print(res);
cat(res, sep="\n");
```
Index

* datasets
  opts, 13
  apply_graph_theme, 2
  asrt (source), 18
  assert (source), 18

  base30and36conversion
    (base30toNumeric), 3
  base30toNumeric, 3
  base::readLines(), 9, 10
  base::stop(), 7
  base::strsplit(), 10
  base::warning(), 7

  c.justifierElement (source), 18
  cat, 4, 5
  cat0, 4
  clean_workspace, 5

  dcsn (source), 18
  dcsn(), 12
  decide (source), 18
  decision (source), 18
  DiagrammeR::DiagrammeR, 2
  DiagrammeR::render_graph(), 15

  export_justification, 5
  export_justification(), 17

  generate_id, 6
  generate_id(), 4
  get (opts), 13
  get_workspace, 7
  get_workspace_id, 8
  ggplot2::ggplot(), 10
  graphics::plot(), 15
  gsub(), 16

  ifelseObj, 8

  jstf (source), 18
  jstf(), 12
  justify (source), 18

  load_justifications, 9
  load_justifications(), 14
  load_justifications_dir
    (load_justifications), 9
  log_decision, 11

  merge_specLists, 12

  numericToBase30 (base30toNumeric), 3
  opts, 13

  parse_justifications, 14
  plot.justifications
    (load_justifications), 9
  plot.justifierDecisionGraph
    (parse_justifications), 14

  print.justifications
    (load_justifications), 9
  print.justifierDecisionGraph
    (parse_justifications), 14

  readLines(), 10
  repeatStr, 16
  repStr (repeatStr), 16
  reset (opts), 13

  sanitize_for_DiagrammeR, 16
  save_workspace, 17
  set (opts), 13
  set_workspace_id, 18
  source, 18
  srce (source), 18
  strwrap(), 25

  to_specList, 22

  vecTxt, 23
vecTxtQ(vecTxt), 23
workspace, 24
wrapVector, 25

yum::load_and_simplify(), 14
yum::load_yaml_fragments(), 10