Package 'justifier'

September 18, 2021

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

Version 0.2.2

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, jsonlite (>= 1.7), 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 2021-09-18 04:10:02 UTC

R topics documented:

`apply_graph_theme` *Apply multiple DiagrammeR global graph attributes*

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

```
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)
```

`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)

numericToBase30(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));
```

`c.justifierElement` *Programmatically constructing justifier elements*

Description

These functions can be used to programmatically construct justifications.

Usage

```
## S3 method for class 'justifierElement'
c(...)

## S3 method for class 'justifierStructured'
c(...)

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,
  ...
)
```

Arguments

`...`	Additional fields and values to store in the element.
`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).
`xdoi`	For `sources`, XDOI identifier (a DOI, or, if that does not exist, ISBN or other unique identifier of the source).
`source`	In assertions, the source (or sources) that the assertion is based on can be specified using `srce()`.
`assertion`	In justifications, the assertion (or assertions) that the justification is based on can be specified using `asrt()`.
`alternatives`	The alternatives that were considered in a decision.
`justification`	In decisions, the justification (or justifications) that the decision is based on can be specified using `jstf()`.

Value

The generated object.

Examples

```
### Programmatically create a partial justification object
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
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
```

```
justifierJustification;

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

### Programmatically create a justification object for a full decision
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
twoDecisions <-
  c(simpleDecision,
    fullJustifierObject);

### Show the combination
twoDecisions;
```

`c.justifierStructuredObject` *Concatenate two or more structured justifier objects*

Description

Concatenate two or more structured justifier objects

Usage

```
## S3 method for class 'justifierStructuredObject'
c(...)
```

Arguments

`...`	Structured justifier objects

Value

Invisibly, the concatenated list

`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], "'.");
```

---

clean_workspace *Clean your workspace*

Description

Clean your workspace

Usage

```
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

```
### 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);
```

`export_justification` *Export justification as YAML*

Description

Export justification as YAML

Usage

```
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

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

### Export to YAML
justifierYAML <-
  justifier::export_justification(
    justifierObject,
```

```
    file=NULL);

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

`export_to_json` *Export a justifier specification to JSON*

Description

Export a justifier specification to JSON

Usage

```
export_to_json(x, file = NULL, wrap_in_html = FALSE)

## S3 method for class 'justifierStructuredObject'
export_to_json(x, file = NULL, wrap_in_html = FALSE)

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

Arguments

`x`	The justifier specification.
`file`	Optionally, a file to save the JSON to.
`wrap_in_html`	Whether to wrap the JSON in an HTML element.
`...`	Any additional arguments are ignored.

Value

If a file is specified to write, to, `x` will be returned invisibly to allow building a pipe chain; if `file=NULL`, the resulting JSON will be returned as a character vector.

Examples

```
### Programmatically create a justification with two assertions
### but without sources; flatten it; and show the json
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
) |>
  justifier::flatten() |>
  justifier::export_to_json();
```

`flatten` *Flatten a justifier tree*

Description

Flattening takes all justifications, assertions, and sources from their parents and returns a structured justifier object containing these elements in separate lists, with each occurrence replaced with a reference to the corresponding identifier.

Usage

```
flatten(x, ..., recursionLevel = 0, silent = justifier::opts$get("silent"))

## S3 method for class 'multipleJustifierElements'
flatten(x, ..., recursionLevel = 0, silent = justifier::opts$get("silent"))

## S3 method for class 'singleJustifierElement'
flatten(x, ..., recursionLevel = 0, silent = justifier::opts$get("silent"))
```

Arguments

`x`	The justifier object or objects.
`...`	Additional arguments are passed to the methods.
`recursionLevel`	The depth of the recursion
`silent`	Whether to be silent or chatty

Value

A flattened justifier object.

Examples

```
### Programmatically create a justification with two assertions
### but without sources
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
  );

### Flatten it into a structures justifier object
structuredJustification <-
  justifier::flatten(
    justifierJustification
```

```
  );

### Check it
str(structuredJustification, 1);
```

`generate_id` *Generate unique identifier(s)*

Description

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

Usage

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

Arguments

`type`	The type of the justifier object; `D`, `J`, `A` or `S`.
`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).

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	Whether to be suppress messages.

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

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

Arguments

silent	Whether to be suppress messages.

Value

Invisibly, the workspace identifier.

Examples

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

idRef *Create a reference to one or more justifier objects*

Description

Create a reference to one or more justifier objects

Usage

```
idRef(x, what = NULL, silent = justifier::opts$get("silent"))

## S3 method for class 'singleJustifierElement'
idRef(x, what = NULL, silent = justifier::opts$get("silent"))

## S3 method for class 'multipleJustifierElements'
idRef(x, what = NULL, silent = justifier::opts$get("silent"))

## S3 method for class 'justifierIdRef'
idRef(x, what = NULL, silent = justifier::opts$get("silent"))

## S3 method for class 'character'
idRef(x, what = NULL, silent = justifier::opts$get("silent"))

## S3 method for class 'justifierStructured'
idRef(x, what = NULL, silent = justifier::opts$get("silent"))
```

Arguments

x	The identifier(s)
what	Optionally, what x is (decision, justification, assertion, or source).
silent	Whether to be silent or chatty.

Value

The justifier id reference object.

Examples

```
exampleSource <-
  justifier::source("This is a book about R.");

exampleAssertion <- justifier::assert(
  "R is a functional language",
  source = justifier::idRef(exampleSource)
);

### Get and show the reference
```

```
(sourceId <- exampleAssertion$source);

sourceId <- as.character(sourceId);

### Manually assign an identifier
justifier::idRef(sourceId);

### Repeat while specifying what we're passing
justifier::idRef(sourceId, what="source");
```

`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

```
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

Examples

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

`import_from_json` *Import a structured justifier object from JSON*

Description

Import a structured justifier object from JSON

Usage

```
import_from_json(x)
```

Arguments

`x`	Either a path to an existing file, or a character vector with the JSON to import.

Value

The justifier object.

Examples

```
### Programmatically create a justification with two assertions
### but without sources; flatten it; and show the json
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
) |>
  justifier::flatten() -> originalObject;

originalObject |>
  justifier::export_to_json() ->
  exportedJSON;

### And import it again
importedFromJSON <-
  justifier::import_from_json(
    exportedJSON
  );
```

load_justifications *Load Justifications from a file or multiple files*

Description

These function 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 justifications 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 justifications.

`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 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`,

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

```
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:

---
assertion:
  -
    id: assertion_SD_decision
```

```
    label: Sleep deprivation affects the decision making proces.
    source:
      id: source_Harrison
---

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:

---
source:
  -
    id: source_Harrison
   label: "Harrison & Horne (2000) The impact of sleep deprivation on decision making: A review."
    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.
';

justifier::load_justifications(text=exampleMinutes);

### To load a directory with justifications

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

`log_decision` *Document a decision*

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

```
log_decision(
  label,
  description = "",
  alternatives = "",
  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)
```

Arguments

x, y	The two justifier specification lists

Value

A merged justifier specification list.

Examples

```
### Add example
```

`opts` *Options for the justifier package*

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

```
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 common 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.

Examples

```
### 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

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

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

## 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
```

```
names(parsedJustifications);
```

`randomSlug` *Generate a random slug*

Description

idSlug is a convenience function with swapped argument order.

Usage

```
randomSlug(x = 10, id = NULL, chars = c(letters, LETTERS, 0:9))

idSlug(id = NULL, x = 10, chars = c(letters, LETTERS, 0:9))
```

Arguments

`x`	Length of slug
`id`	If not NULL, prepended to slug (separated with a dash) as id; in that case, it's also braces and a hash is added.
`chars`	Characters to sample from

Value

A character value.

Examples

```
randomSlug();
idSlug("identifier");
```

`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);
```

`sanitize_for_DiagrammeR` *Sanitize for DiagrammeR*

Description

Basically a wrapper for `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");
```

save_workspace *Save your workspace*

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 *Set your justifier workspace identifier*

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");
```

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

```
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

```
### 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
class(loadedMinutes["assertion"]);

### 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;
```

---

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));
```

`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

```
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

```
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