Documentation

Petr Pajdla

Documentation

Documentation

Crucial both for other people and for future-you!

- `README` File
- *Object* documentation
  - documentation of individual objects (functions)
  - accessed by `help(myfun)` or `?myfun`
- Vignettes
  - long-form documentation
  - whole *workflows* of functions implemented in the package

Documenting functions

- Documentation goes into the `man/` directory
- *Each* (non-lambda) function should be documented

For `myfun()` function:

```
myfun <- fucntion(x) {
  ...
  }
```

a file `myfun.Rd` exists in `./man`

The .Rd file

- Plain text file, loosely based on LaTeX

```
\name{myfun}
\alias{myfun}
\title{myfun, doing this and that (...)
The myfun function calculates (...)}
\usage{
myfun(x, y, method = "something", type = (...))
}
\arguments{
\item{x}{x is a vector of length (...)}
}
\description{
The myfun function calculates (...)
}
```

roxygen2

roxygen2

- Using **roxygen2** simplifies the process of creating documentation

Workflow

1. Create a **function**
2. Add ***roxygen* comments** in the `.R` script with special *tags*
3. Run `roxygen2::roxygenize()` to generate documentation (`devtools::document()` does the trick as well)
4. **Preview** the documentation (`help()` or `?`)
5. Have fun and **repeat**!

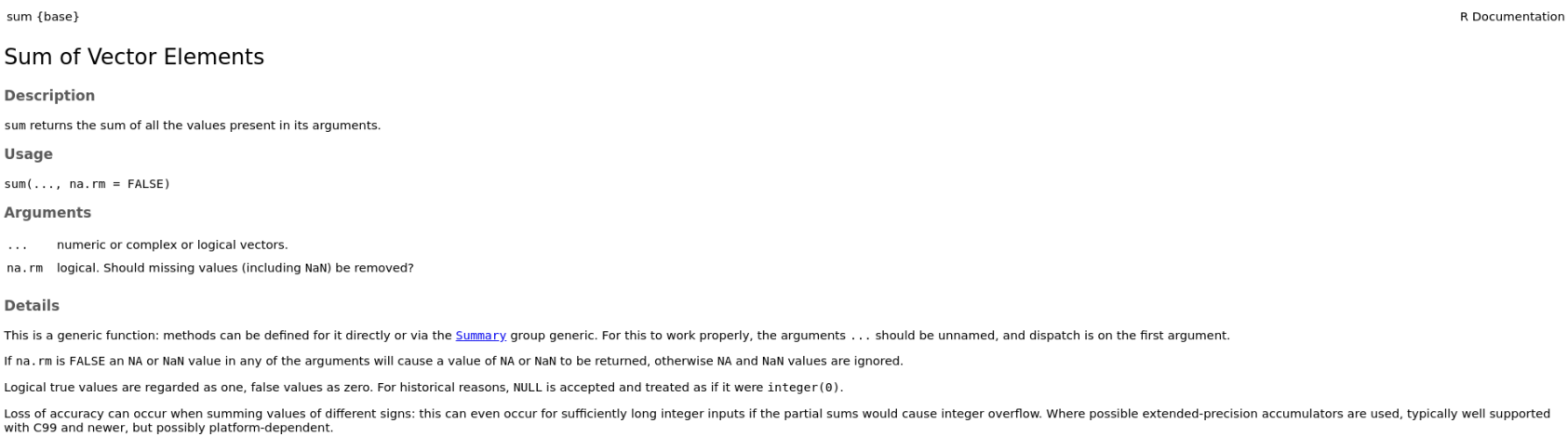
roxygen comment blocks

- Written *above* the function definition in a given `.R` file
- Always start with `#'`
- Tags for various *sections* `#' @tag`

```
#' My Function.
#'
#' My function does this and that.
#'
#' @param x (...)
#' @return A number (...)
#'
myfun <- function(x) {
  ...
}
```

Basic structure

- First sentence = **title of documentation**
  - Sentence Case, ends with a Full Stop.
- Second paragraph = **description**
  - short description of the function
- Third and subsequent paragraphs = *details section*



sum {base} R Documentation

Sum of Vector Elements

Description

sum returns the sum of all the values present in its arguments.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

...	numeric or complex or logical vectors.
na.rm	logical. Should missing values (including NaN) be removed?

Details

This is a generic function: methods can be defined for it directly or via the Summary group generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the first argument.

If na.rm is FALSE an NA or NaN value in any of the arguments will cause a value of NA or NaN to be returned, otherwise NA and NaN values are ignored.

Logical true values are regarded as one, false values as zero. For historical reasons, NULL is accepted and treated as if it were integer(0).

Loss of accuracy can occur when summing values of different signs: this can even occur for sufficiently long integer inputs if the partial sums would cause integer overflow. Where possible extended-precision accumulators are used, typically well supported with C99 and newer, but possibly platform-dependent.

Figure 1: part of documentation of sum function

Tags

Parameters (function arguments)

`@param name description`

- function parameters - document all inputs!
- A sentence, paragraph or even longer text if necessary.

```
#' @param x A numeric vector.
#' @param data A data frame. See below for details.
#' @param x,y Numeric vectors.
```

Examples of code

`@examples`

- `R` code with examples of the function in practice
- the code **must** work, it is run during the checks (`R CMD check`)
- `\dontrun{...}` = code is not run

`@example`

- contains relative path to `.R` file with examples

```
#' @examples
#' mean(c(1, 2, 3))
#'
#' \dontrun{
#'   mean(c("a", "b", "c"))
#' }
```

Function output

`@return description`

- describes the output of the function

```
#' @return The default method returns a length-one
#'    object of the same type as \code{x}. If (...)

#' @return An object of the same type as \code{data}
#'    is returned.
```

Export the function

`@export`

- exports the function for the *end* user
- adds a proper line to **`NAMESPACE`** file
- functions that are not exported remain *internal*
- to access internal function, use `pckgname:::funname`

Linking

`@seealso`

- points to other resources inside the package or elsewhere

```
#' @seealso For details, see similar
#'    function \code{\link{funname}}.

#' @seealso See \url{http://...} for details.

#' @seealso See \code{\link[pckgname]{funname}}
#'    function from package (...)
```

Other tags

`@section`

- allows to break long texts, i.e., in *Details* section

`@aliases alias1 alias2 ...`

- adds additional aliases to the function
- the topic is found by `?alias1` or `?alias2` etc.

Rd Markup

Formatting

- `@` = start of roxygen tag, to write at sign (@), use `@@`
- % = LaTeX comment sign, escape it by backwards slash `\%`
- `\` = LaTeX escape sign, escape it to get a single bw. slash `\\`

```
#' @author My Name <my.email@@fuu.bar>
```

- `\code{}` - code snippets
- `\link{funname}` - link to function in this package
- `\link[pckgname]{funname}` - link to function in another package

```
#' @return Object of class \code{data.frame}.
#' @seealso See \code{\link[base]{data.frame}} for details
```

Equations

- standard LaTeX math (without AMS or other extensions)
- `\eqn{}` - inline equation
- `\deqn{}` - block (*display*) equation

```
#' @section Details on maths
#' Sample mean is calculated as
#' \deqn{\overline{x} = \frac{1}{n}\sum_{i=1}^n x_i}
```

should result in something like this:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

(but in the html help files, equations are simplified...)

Wrap up & exercise

Where to get help?

- Do I have to know all of this by hearth?!
  - Luckily, nope!
- package development cheat sheet (https://www.rstudio.com/resources/cheatsheets/)
- See the introduction and other vignettes for `roxygen2` package
  - `vignette("roxygen2")`
  - `vignette(package = "roxygen2")`

If you are using `RStudio`:

- go to `Help/Roxygen Quick Reference`

Exercise

- Do you have a package with two functions defined?
  - `doublemean()`
  - `normalmean()` (?)
- Let's document the functions!
  1. Add meaningful title
  2. Add basic description
  3. Document the parameters
  4. What does the function return?
  5. Any examples?
  6. Export the function to the `NAMESPACE`
  7. Add a link to function `mean` from `base R`
  8. Run `devtools::document()`
  9. Explore the help files with `?doublemean`