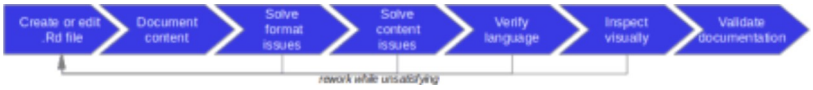
If your still hand-crafting your *R* documentation files, then you face a time consuming activity. Trying to foresee time needed to get a documentation to an expected quality level stays a real challenge.

The generic process to create an *R* documentation file looks like



Whatever approach you use, traditional one using *base R*, or more modern ones like *roxygen2*, this process requires iterations, and it is generally where it hurts to work under scheduled time while guaranteeing the right level of documentation quality. Not only the process is iterative, it is also generally poorly industrializable.

Another approach is offered by package *wyz.code.rdoc*, that is generating documentation from code, **by code**. This allows much easier and deeper industrialization level. It permits to create quite easily pieces of *R* code to generate any *R* documentation parts or the whole documentation, according to your needs. Let's see an example.

Produce documentation from code

Imagine, you need to create a documentation for following *dummy R* function that resides in a package named *lambdaPackage*.

```
specialAddition <- function(x, ...) suppressWarnings(sum(unlist(list(x, ...)),
0L))
```

Using *wyz.code.rdoc*, here is the way to create your documentation

```
library(wyz.code.rdoc)

# your documentation examples to consider
examples <- list(
  function() { specialAddition(1, 2, 3) },
  function() { specialAddition(1:3, 1:7, 1:11) },
  function() { specialAddition(as.list(1:13),
                               c(1:4, Inf, -Inf, NaN),
                               runif(13, 0, 19)) }
)

# your documentation complementary parts to consider
shortcuts <- shortcuts(formalArgs(specialAddition), FALSE)
postprocessor <- shortcuts(formalArgs(specialAddition), TRUE)
pc <- ProcessingContext(
  extraneous_l = list(
    title = 'Special Addition',
    description = sentensize('add vector or list of numerics whatever',
```

```
                              'their length'),
    value = sentensize(paste('a single', shortcuts$type$numeric, 'value')),
    details = paste('Classical', shortcuts$doc$r,
                    'warning messages about recycling are managed internally',
                    'and discarded by the function.'),
    examples = convertExamples(examples, captureOutput = FALSE)
  ),
  postProcessing_l = list(
    arguments = function(content_s) {
      sub('XXX_00\\d{1}', paste('a', postprocessor$type$vector, 'or',
postprocessor$type$list,
                                'of', postprocessor$type$numeric, 'values'),
          content_s, perl = TRUE)
    },
    author = function(content_s) NULL # remove authorship
  )
)

# The generation of the manual page
p <- produceManualPage(InputContext(NULL, 'specialAddition', 'lambdaPackage'),
                       pc, GenerationContext('~/tmp', overwrite = TRUE))

## File ~/tmp/specialAddition.Rd passes standard documentation checks
```

Generation process overview

Quite simple, as the process is straightforward

1. provide your examples as a list of functions taking no parameter
2. set a processing context, expressing sections to add, and any required post processing actions to get fully documented manual page,
3. generate your *R* manual page using function *produceManualPage*
4. Get immediate result of compliance against *R* documentation checks

Notice some important points

1. Here, generated manual page
    1. is produced for a package and package name matters in such context,
    2. took into consideration your content and format customizations,
    3. has used the 3 examples you provided,
    4. patched at your demand the content of function arguments,
    5. removed at your demand authorship from the output
2. Function *shortcuts* allows easy and convenient reuse of very common *R* documentation language markup sequences. You don't have to learn these markup sequence, just have to known how to reuse those pieces.
3. Produced manual page has gone through standard *R* documentation check tool to ensure documentation validity and to provide an immediate feedback.

Preview

Preview captured from *RStudio*.

Try it and become a fan

Main benefits of proposed approach are

1. Nearly full coverage of the previously exposed *R* process
2. High flexibility customization
3. Ease of use
4. Reuse enabling
5. Better documentation quality production
6. Higher individual productivity

Give *wyz.code.rdoc* package a try as it will increase your *R* manual page creation productivity. If you create packages, you have to provide manual pages for each function you publish, you will thank yourself achieving this task in less time, with better quality of result, in a possibly fully automated and reproducible way.

Watch for next post on *wyz.code.rdoc* for more advanced features to use in *R* documentation generation.