Rmarkdown conditional chunks to create multilingual pdf and html with images

Sébastien Rochette - StatnMap

21 mars, 2017

Contents

⬛ This version is the english version.

Go back on my website:

StatnMap
Courses and consulting

[HOME]() | [BLOG]() | [TUTORIALS]() | [CONTACT]()

This page is presented here on my website

---

1.   Knit with multiple rendering options

I want to be able to knit my rmarkdown documents in different speaking language (english and french). I also want them to give similar rendering in html and in latex pdf so that I can use them in my courses as html version on the laptop, but also as printed version through the latex pdf. By the way, I also have a teacher and a student version for the R-scripts that I can purl from the Rmd file.

This means many options for a **unique** rmarkdown document. Using a unique document allow me to directly modify things for the different versions at the same place. I do not have to jump from a file to another and verify that I copied every modification in the R-script or other. . .

I personally use an external *"run-all"* R-script to choose options to run the different versions and rename output files.

1.1. A non exhaustive list

In this article, I list some of the tips I have to use to make this work and some bonus:

- Use chunks options to render text according to speaking language chosen,
- Use functions to render text color, background or format similar in pdf and html outputs,
- Print inline (or not) images depending on speaking language,
  - Render two images side by side and centered on the page for html and pdf,
- Allow verbatim chunk environment with background color for html and pdf,
- Output R-script included in chunks for tutorials, with different versions for teacher and students.

I will modify this list if I face new difficulties or I find better ways to do things.
Files needed to create this article are stored here on my github.

```r
packageVersion("knitr")
```

```
## [1] '1.15.1'
```

1.2. Functions for html or pdf specific rendering

Because some text and background colors are not written in the same way for html and lateX outputs, I need to use functions that will add either html formatting or latex formatting:

- `colFmt` is used directly with color names like blue, red, ...
- `styleFmt` is used in combination with definition of a style in external files.
- `beginStyleFmt` and `endStyleFmt` are a separated variante of `styleFmt` to be used if your text contain special characters (like underscore) that may be misinterpretated by lateX.
  Option `type` can be defined to use `div` instead of `span` to allow multiple lines with html output. Similarly for latex, if `type == "div"` is defined, command will be used with `begin` and `end`.
- `beginVerbatim` and `endVerbatim` are specific to embed verbatim code text with background color, which is not implemented by default in lateX.

```r
# Simple color Format functions
colFmt <- function(x, colorname, type = "span") {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer'))
    paste0("\\textcolor{", colorname, "}{", x, "}")
  else if (outputFormat == 'html')
    paste0("<", type, " class='", colorname, "'>", x, "</", type, ">")
  else
    x
}
# Style format function, not specific for textcolor
styleFmt <- function(x, textstyle, type = "span") {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer'))
    paste0("\\", textstyle, "{", x, "}")
  else if (outputFormat == 'html')
    paste0("<", type," class='", textstyle, "'>", x, "</", type, ">")
  else
    x
}
# Style format options for longer text output
beginStyleFmt <- function(textstyle, type = "span") {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    if (type == "div") {
      cat("\\nopandoc\\begin{", textstyle, "}}\n", sep = "")
```

```r
    } else {
      paste0("\\", textstyle, "{")
    }
  } else if (outputFormat == 'html') {
    if (type == "div") {
      cat("<", type, " class='", textstyle, "'>", sep = "")
    } else {
      paste0("<", type, " class='", textstyle, "'>")
    }
  } else {
    ""
  }
}
endStyleFmt <- function(textstyle, type = "span") {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    if (type == "div") {
      cat("\n\\nopandoc{\\end{", textstyle, "}}", sep = "")
    } else {
      paste0("}")
    }
  } else if (outputFormat == 'html') {
    if (type == "div") {
      cat("</", type, ">", sep = "")
    } else {
      paste0("</", type, ">")
    }
  } else {
    ""
  }
}
# Specific to verbatim environment
# to be able to show you the 'asis' chunks with background color.
beginVerbatim <- function() {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    cat("\\begin{blueShaded}\n\\begin{verbatim}\n")
  } else if (outputFormat == 'html') {
    cat("<div class='blueShaded'>\n```\n", sep = "")
  } else {
    ""
  }
}
endVerbatim <- function() {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer')) {
    cat("\\end{verbatim}\\end{blueShaded}")
  } else if (outputFormat == 'html') {
    cat("\n```\n</div>", sep = "")
  } else {
    ""
  }
}
```

The definition of different styles are in external lateX header file for pdf outputs and should be called in
the YAML header of your rmarkdown file `in_header:  header.tex`. This lateX file should include the
definition of all colors and format to be used with the previous functions. For instance, here:

```
\usepackage{color}
\definecolor{blueSeb}{RGB}{30,115,190}
\definecolor{advertcolor}{HTML}{FF8929}

\definecolor{backcolor}{RGB}{235, 235, 235}
\newcommand{\mybox}[1]{\par\noindent\colorbox{backcolor}
  {\parbox{\dimexpr\textwidth-2\fboxsep\relax}{#1}}}

\newcommand{\advert}[1]{\textit{\advertcolor{#1}}}
\newcommand{\codecommand}[1]{\texttt{\colorbox{backcolor}{#1}}}
```

By the way, pandoc do not consider text between `begin` and `end` lateX entries as markdown syntax. To be able to consider your markdown syntax when using functions `beginStyleFmt` and `beginStyleFmt` you need to define a command in your `header.tex` file as follows:

```
\newcommand{\nopandoc}[1]{#1}
```

Similarly, for html rendering, functions should be used in an external css file with option `css: style.css` in the YAML header of your rmarkdown file. Here for instance the css file includes:

```
.advert {
  color: #FF8929;
  font-style: italic;
}
.codecommand {
  background-color:#E0E0E0;
  font-family: Courier New, Courier, monospace;
}
.Shaded {
  background-color:#E0E0E0;
  font-family: Courier New, Courier, monospace;
}
```

2.   Choose a language to knit

If you want to be able to knit your documents in different language without using two different rmarkdown files, here are my tips.

```
```{r R-Lang, echo=TRUE, eval=FALSE}
# Choose the language at the beginning of your script or knit from external file
lang <- c("EN", "FR")[1]
```
```

2.1.   Embed text in 'asis' chunks

In 'asis' chunks, with option `echo=TRUE`, you can write rmarkdown syntax which produces text (partially) in the correct format. The aim is to use the command `eval` to render the text according to the chosen language:

```
<!-- english -->
```{asis, eval=(lang == "EN"), echo=TRUE}
### For titles and subtitles when at the beginning
This above chunk allow for markdown syntax


### And for titles in the middle if there is space line above
```

```
And you can use it for normal text or _italic_ or **other**
```

<!-- french -->
```{asis, eval=(lang == "FR"), echo=TRUE}
### Pour les titres et sous-titres au début du "chunk"
Le "chunk" ci-dessus permet d'utiliser la syntaxe markdown

### Et les titres en milieu de "chunk" s'il y a une ligne vide avant
Et vous pouvez utiliser le texte normal, _italic_ ou **autre**
```
```

2.1.1   For titles and subtitles when at the beginning

This above chunk allow for markdown syntax

2.1.2   And for titles in the middle if there is space line above

And you can use it for normal text or *italic* or **other**

2.2.   Include R outputs in conditional chunk

```
```{r CodeInChunkEN, echo=(lang == "EN"), eval=(lang == "EN"), results='asis'}
cat(
'### Example of title in a chunk
#### Example of subtitle in a chunk
- ', styleFmt("cat", "codecommand"), ' should be used with ',
styleFmt("sep = \"\"", "codecommand"), ' otherwise a space will be added
before "##subtitle", which avoid it to be recognised as a subtitle in markdown.
- ', styleFmt("eval", "codecommand"), ' can be used with R object to knit
chunk specific to language. Here ', styleFmt("lang", "codecommand"),
' is ', lang,  '\n',
'- Double space should work to define new paragraph, but when using R
functions, it may require "\\\\n"
_(Note the number of backslash in the code here to print "\\\\n" correctly...)_.
    + ', styleFmt("For instance when using the format function,
                following two spaces won\'t work:", "advert"), '  ',
'    + This sentence won\'t be on a new line.',
sep = "")
```
```

2.2.1   Example of title in a chunk

2.2.1.1   Example of subtitle in a chunk

- `cat` should be used with `sep = ""` otherwise a space will be added before "##subtitle", which avoid it to be recognised as a subtitle in markdown.
- `eval` can be used with R object to knit chunk specific to language. Here `lang` is EN
- Double space should work to define new paragraph, but when using R functions, it may require "\n" *(Note the number of backslash in the code here to print "\n" correctly. . .)*.
  - *For instance when using the format function, following two spaces won't work:* + This sentence won't be on a new line.

2.3. Conditional images and path

2.3.1 When images are in a defined directory and are shown in paragraph different that text
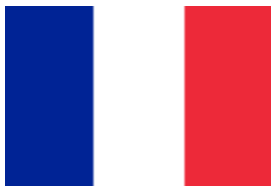
```
```{asis, eval=(lang == "EN")}
![](./myfigdirectory/gb.png) Directory for english version
```
```{asis, eval=(lang == "FR")}
![](./myfigdirectory/fr.png) Fichier pour la version française
```
```

2.3.2 When images are in a changing directory and are shown inline

Only code of the last point is shown here, but you can find the others in the raw Rmarkdown file used to create this page.

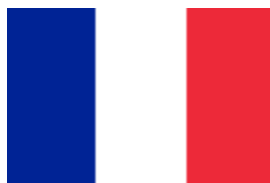```
figWD <- paste0("./myfigdirectory", lang)
```

-  Image without condition
-  No condition but allow to resize image (if `out.width` is not set in general knitr options).

```
  - `r
    if (lang == "EN") {
      knitr::include_graphics(paste0(figWD, '/gb.png'), dpi = 400)
    } else if (lang == "FR") {
      knitr::include_graphics(paste0(figWD, '/fr.png'), dpi = 400)
    }`
    `r
    if (lang == "EN") {
      "Conditionnal image: English flag shown if 'lang' is 'EN'"
    } else if (lang == "FR") {
      "Image avec condition : Le drapeau français est affiché si 'lang' est 'FR'"
    }`
```

- Conditionnal image: English flag shown if 'lang' is 'EN'

3. Two images side-by-side and centered

A problem while using two images side by side using `"knitr::include_graphics"` in the same chunk is that option `fig.align = 'center'` applies separately to the two images, so that there are one under the other, in two different paragraphs:

Here, we use two functions to be place before and after the chunk with `results='asis'` and a space between images:

```r
beginCentering <- function() {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer'))
    cat("\n\\nopandoc{\\begin{center}}\n", sep = "")
  else if (outputFormat == 'html')
    cat('\n<p style="text-align: center">\n', sep = "")
  else
    ""
}
endCentering <- function() {
  outputFormat <- knitr:::pandoc_to()
  if (outputFormat %in% c('latex', 'beamer'))
    cat("\n\\nopandoc{\\end{center}}\n", sep = "")
  else if (outputFormat == 'html')
    cat("\n</p>\n", sep = "")
  else
    ""
}
```

The chunk is then written as follows:

```
```{r, echo=FALSE, out.width='25%', fig.align='default', results='asis'}
beginCentering()
knitr::include_graphics(paste0(figWD, "/fr.png"))
cat("\\vspace{1cm}")
cat(" ")
knitr::include_graphics(paste0(figWD, "/gb.png"))
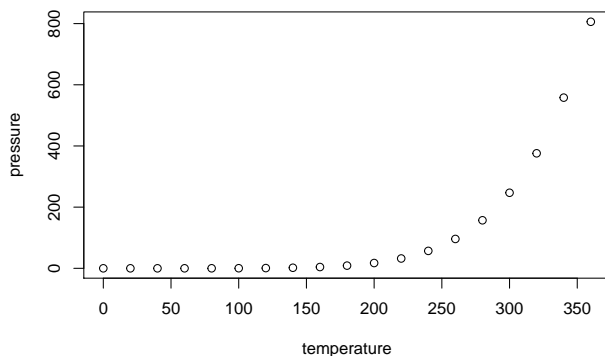endCentering()
```
```



# 4.   Student and teacher versions

## 4.1.   Create the tutorial roadmap

This one is pretty easy. You only have to set your `purl` option depending if your chunk is for student or teacher. I personally use `eval=TRUE` with the teacher version that should produce the result targeted by students. And I use `eval=FALSE` with the student version because this version should not work.

### 4.1.1  Exercise example

Reproduce the following figure from the "cars" dataset.

```
##       speed           dist
##  Min.   : 4.0   Min.   :  2
##  1st Qu.:12.0   1st Qu.: 26
##  Median :15.0   Median : 36
##  Mean   :15.4   Mean   : 43
##  3rd Qu.:19.0   3rd Qu.: 56
##  Max.   :25.0   Max.   :120
```


4.2. Purl the rmarkdown file

If you purl this rmarkdown file using the "teacher" option, you'll get this R script:

```r
# A script for teacher with solutions -----------------------------------------
summary(cars)
# Plot pressure against temperature
plot(pressure)
```

And this R script if you purl with the "student" option:

```r
# A script for students without solutions -------------------------------------
summary(cars)
# Plot pressure agains temperature
plot(...)
```

5. Try this script !

All files necessary to produce this page are available on my github. All outputs in french and english, in pdf and html, as well as student and teacher R-script are also in the repository. Use the *"Run_all_and_choose_options.R"* file to produce them on your own.

5.1. Some limits

A limit to the use of "\begin" and "\end" environment with lateX is that pandoc do not convert markdown syntax included in that environment. It is read as verbatim. Maybe, I'll find a way to get rid of that.
One of the drawback of this kind of multilingual rmarkdown script is that code highlighting is messy in Rstudio... This specific one in very messy because I wanted to show you the complete chunks in the output file, but it is quite difficult to do it in a clean way as "asis" chunks do not allow for "echo" and "verbatim" options. If you have any better way to do it, you can comment here or in the github repository itself.
By the way, if your file starts to be too big like the present one, do not hesitate to work with child Rmd files.

6.   Supplemental tips

LateX default template of rmarkdown load several packages and defines several parameters to format the final text output. If you do not want to create your own template, you can define, like here a tex file to load `"in_header"`.

However, you may encounter some error messages due to incompatible options. In the "header_tips.tex" file provided with the present templates, you will find different modifications of the pdf template:

- Find which language is loaded with lateX package babel and define new commands according to language chosen. This is useful here because we have a multilingual template that uses parameter `lang` in the YAML, which I parameterized according to language to knit.

- Formatting section titles to be colored or defined as you want with package "titlesec" requires to unload section format that rmarkdown provided and revert it back.

- Modify background color of rmarkdown verbatim box.

More tips on R, models and spatial things on my website

This page is presented *here on my website*

StatnMap
Courses and consulting