stationery Package Overview: Guides, Slides, and Reports

Paul E. Johnson, Director, CRMDA

May. 4, 2018

Abstract

This document surveys methodologies for literate document preparation. It offers an overview for 8 document templates provided by the R package named `stationery`. The document compares LaTeX-based strategies (using the code chunk engines Sweave and knitr) with documents prepared with markdown. We discuss strengths and weaknesses of each method and then offer some advice about how authors should plan their projects.

1 Introduction

Researchers who have been using word processors to prepare their research articles are understandably bewildered by recent talk about reproducible research documents (Leisch, 2002; Xie, 2015; sto, 2013; Xie, 2016). The banner of "reproducible documents" hangs over scholars who combine code and analysis into a single authoring process. We are in a new era of emphasis on reproducible research and the integration of statistical analysis with report preparation is intended to enhance integrity and reduce the frequency of typographical (and other) errors.

Still, the majority response from most professors in the university seems to echo the exclamation of one university administrator: "I have no idea what that means!" In the *old way* of doing things, one prepares a document in a word processor program. While doing that, one types tables and pastes in graphs. One often types in a bibliography if the word processor does not integrate well with a bibliography manager. The old way is practiced by most social scientists and humanists today.

In the *new way* of doing things, we try to avoid typing tables or pasting in graphs. We want the analysis software to prepare article-ready tables and graphs that can be incorporated into the generation of a final report. In the ideal case, an entire article, lecture, or book can be generated in one single execution that conducts analysis, saves graphics, and assembles them together in a single output document. This is roughly along the lines of the "literate programming" movement started by computer scientist Donald Knuth (1984; 1989), who also created the TeX document preparation system (1984), an accomplishment so massive that its impact on academic research can scarcely be comprehended. The statistical program R (R Core Team, 2017) built its documentation framework on the literate programming philosophy, integrating Knuth's concepts of "weaving" and "tangling" to produce documents and extract code files (Leisch, 2002).

Because researchers in mathematics, engineering, and natural sciences were more likely to be familiar with TeX, and its successor LaTeX, they were more amenable to concepts of reproducible

1425 Jayhawk Blvd.
Watson Library, Suite 470
Lawrence, KS 66045-7594

Web: `https://crmda.ku.edu`
Email: `crmda@ku.edu`
Phone: 785-864-3353

document preparatoin. For social scientists and humanists, where LaTeX was unfamiliar, the idea of integrating code and document preparation did not seem tractable.

Then comes a rude shock. Immediately after one comprehends the general idea of "code embedded within documents," then one is immediately slapped in the face by another grim reality: *There are several competing formats for doing this kind of work.* Furthermore, no single approach is best for all eventualities. One can learn to live within one framework, and accept its limitations, or cultivate some experience with several methods, in danger of remaining a jack of all trades (and master of none).

Some of the impediments that have made LaTeX document preparation difficult have been addressed over the years by the growth of the graphical user interface named LyX (http://www.lyx.org). This is a user-friendly with a look-and-feel of more common word processors, but it generates LaTeX documents that are then compiled by standard LaTeX software. When I say "LaTeX document," I generally mean a document prepared with LyX and then exported to LaTeX. LyX is not a new program. I've been using LyX for 20 years to prepare course notes and slides. It is appealing because it is now available for all major operating systems. It is an open framework and it is also an free/open source software project.

2 What Do You Get with stationery?

First, there document templates. Each document type is delivered in a fully working format that can be immediately compiled into a presentable document. Each one is delivered with an instructional guide that explains how to create a new skeleton file for editing and how to compile the document.

Second, there are R functions that create new templates and assist in compiling them. Each template includes a compiler script, so that if one is partial to using the "command line" to handle repetitive jobs, then this will offer a dependable way to compile documents (and debug problems with them). It is not necessary to use the command line, however, as one can also compile the document using R functions. The LaTeX-based templates can also be compiled within LyX.

Third, there are vignettes, the first of which is the present document.

1. stationery Package Overview: The package overview (this document)
2. Code Chunks
3. Rmarkdown
4. HTML Special Features

`stationery` package document templates

The document types in the `stationery` package, and the labels we use for them, are provided in Table 1. There are several types of documents because authors will make different choices about the document format and the tools covert the document into a presentable format. The formats in which documents are prepared are referred to as "frontends"[1] while the output formats–the ones intended for readers–are the "backends".

[1] In this context, it is a bit vague to say *front end* because each document is converted through several formats in the compilation process. Any intermediate format that precedes another might be referred to as a frontend.

Table 1: Document Types in Stationery

	Template Label	Frontend	Backend	Format	Code Chunk Engine
1	rmd2html-guide	Markdown	HTML	Guide	knitr
2	rnw2pdf-guide-knit	Markdown	PDF	Guide	knitr
3	rnw2pdf-guide-sweave	LaTeX/LyX	PDF	Guide	Sweave
4	rmd2pdf-report	Markdown	PDF	Report	knitr
5	rnw2pdf-report-knit	LaTeX/LyX	PDF	Report	knitr
6	rnw2pdf-report-sweave	LaTeX/LyX	PDF	Report	Sweave
7	rnw2pdf-slides-sweave	LaTeX/LyX	PDF	Slides	Sweave

In Table 1, readers will note that the templates have three-part names like "`rnw2pdf-guide-sweave`". The first part of the label an abbreviation for *frontend2backend.* The nicknames "rnw" and "rmd" represent the two primary frontends used in `stationery`. The two backends are portable document format (PDF) and hypertext markup language (HTML). By mixing and matching the front ends and the backends, we have identifiers like "rnw2pdf" and "rmd2pdf". To avoid creating additional labels when it is not completely necessary, we use the label "rnw" to refer to documents that are prepared either as Rnw or LyX files (Since we use LyX to generate Rnw, this is not a harmful simplification).

The middle part of the format label is either "guide", "report" or "slides". These will be discussed in the next section.

The final part of the template label represents the engine that is used to convert the embedded computer code into statistical output. This might be either "sweave" or "knit". These names refer to the code chunk engines, Sweave (Leisch, 2002) and knitr (Xie 2014, 2018). Sweave and knitr are two, separate, not-entirely-compatible code frameworks. A possible source of confusion is that Sweave, which is the original engine, uses the term "Sweave" (an homage to Knuth's term "weave") to refer to the act of processing S or R code chunks in a LaTeX document, while the knitr package introduced a new term for this, "knit".

The markdown document movement is associated with the John Gruber's Daring Fireball website (https://daringfireball.net/projects/markdown) and John MacFarlane's ambitious software program `pandoc` (https://johnmacfarlane.net/pandoc). Gruber is credited with the main idea of this movement, which is to allow authors to prepare documents in a readable format which can then be converted into beautiful PDF, HTML, or other formats. pandoc is a Swiss army knife of format conversion and its implementation is what makes most of this actually work.

Because markdown is a movement, rather than a product, it allows the creation of specialized dialects. and its author's webpage () has been an effort to develop an alternative style of document markup that is not so "ugly" as HTML and not so difficult to learn as LaTeX. While the general idea of markdown is quite appealing, the devil is in the details, as usual. A host of competing dialects flowered almost immediately and now it is quite possible to write a document "in" markdown that is not intelligible to a practitioner of another kind of markdown. Here, of course, we are interested mainly in R markdown, a variant of markdown that is associated with the knitr package for R.

In the stationery package we now have 7 working document skeletons that are summarized in Table 1.

Guides versus reports

A **guide** is intended for students or other learners. A guide will generally include code and output excerpts. Preferrably, the codeexamples will have line numbers and syntax highlighting. In most cases, a guide format is not format suitable for reports to clients or for journal articles.

A **report** is a more formal document. A report is suitable for a journal article or a technical report. A report has less (maybe no) code and almost never will it include "raw output" from a computer program. A report includes closer-to-publishable tables and figures. The style for "report" documents is fairly well settled. We have a header with the title, the logo, and author information. On page 1, we have contact information, in the style of a stationary footer. The style for "guide" documents is not well settled. It is one of the problems on our immediate agenda.

The slide format we offer is based on LaTeX Beamer () slide format using a customized theme that we prefer. So far, the only code engine for which we have working examples in Sweave. We have experimented with many slide producing strategies using markdown code and none of them have been completely dependable (some of the challenges are discussed in later sections of this document and our vignette named "/??".

There are too many settings

The 'stationery' package has settings that work for our documents. Our compiler scripts leave open the possibility that users can supply additional command line arguments, arguments that have the same names as our R functions rmd2pdf, rmd2html, rnw2pdf, and rnw2html. Still, the number of options that can govern the production of the document, its transition through several formats, is, well, stupefying.

The list of possible settings is immense partly because each document must be converted through several formats. Consider a document that is written with R markdown. In order to achieve success, the document must undergo these transitions

1. Rmd -> md. R code chunks are isolated, calculated, and replaced by R output. This creates a markdown document.
2. md -> utf8.md. The markdown document's font encoding is homogenized to us Unicode (utf8).
3. utf8.md -> tex. The markdown is converted into LaTeX. This work is handled by pandoc (>>)
4. tex -> PDF. A LaTeX compiler is needed to do this conversion. Perhaps the most frequently used is "pdflatex", but there are others that are increasingly popular and compatible with additional fonts and character sets. If the document includes a bibliographic database, one of the companion programs, such as bibtex, biber, or biblatex must be called on to do its work.

The transition from 'utf8.md' to LaTeX, considered in isolation, is nearly overwhelming. When we compile a markdown document named "crmda.Rmd", we can investigate the details and the 'pandoc' command is the following elaborate request.

```
$ /usr/bin/pandoc +RTS -K512m -RTS crmda.utf8.md
--to latex --from markdown+autolink_bare_uris+ascii_identifiers
   +tex_math_single_backslash --table-of-contents --toc-depth 2 --
   template theme/report-boilerplate.tex --highlight-style haddock
   --latex-engine
pdflatex --listings
```

It is a little bit difficult to design a template that both allows some user customization while maintaining a consistent look and feel. We believe the default configuration in our document packages should ”just work”. However, if specialized adjustment is necessary, we have not made this too difficult for most authors with a “can do” attitude.

3 Usage overview

Here we will illustrate the process of initiating a guide document that is to be prepared with Rmarkdown and will have the HTML backend. Start R in a folder where you would like to create a write-up and run

```
initWriteup("rmd2pdf-report")
```

No directory is specified, so the default setting creates a folder named `rmd2pdf-report`. It will be created in the R current working directory. Within the folder `rmd2pdf-report`, one should find

1. a skeleton template (which the author should rename and edit),
2. an instructional guide (in either PDF or HTML format)
3. a compiler script.

One can specify the directory in which the documents will be written. For example,

```
dirname <- "wherever_you_say"
dir.create(dirname)
initWriteup("rmd2pdf-report", dir = dirname)
```

The output will tell you where the directory ended up, say:

```
[1] "/home/pauljohn/wherever_you_say/rmd2pdf-report"
```

From within R, we can check for a list of files that were created:

```
> list.files(file.path(dirname, "rmd2pdf-report"))
[1] "report-instructions.pdf" "report-instructions.Rmd"
[3] "rmd2pdf.sh"              "skeleton.Rmd"
```

because I want my writeups to collect in the writeup folder of the project.

The first order of business is to rename the subfolder from ”rmd2pdf-report” to something relevant to a writeup. There may be several reports for this project, choose a name that will help you guess which writeup is in which folder. Be aware that ‘initWriteup‘ will erase the files in that folder, if you don’t rename it/them.

Next, we need to make sure the build programs exist and work in your computer. Rename the file ‘report-template.Rmd‘ to match your purpose. My file name is ”crmda.Rmd”.

Test the build tools. Edit your document, insert your name or a title and save the document. Now try to compile it.

The file can be compiled in two ways.

1. Open an R session and use the function ‘rmd2pdf()‘ in the stationery package. I’ll compile my file ‘crmda.Rmd‘.

```
library(stationery)
rmd2pdf("crmda.Rmd")
```

The function `rmd2pdf` is a *wrapper* function. It does not do any real work, it just collects your file name and applies defaults that we have set. It calls the `render` and `pdf_document` functions in the `rmarkdown`. Because `knitr` is the only method for handling code chunks in `Rmarkdown` documents, also involves the `knitr` package.

The `rmd2pdf` function allows the user to add arguments that are passed along to `render` and `pdf_document`. This is explained in the help page for `rmd2pdf`.

2. Run a shell script provided with the template.

It is not necessary to open R to compile the document. The output can also be generated by running a command line script that does the same work.

```
$ ./rmd2pdf.sh crmda.Rmd
```

The rendered output will be a pdf file. The configuration inside `rmd2pdf.sh` indicates that output is written in the current working directory.

If one uses Rstudio as a text editor for an Rmd file, one may not obtain satisfactory output by compiling the document with the pull down menus.

3.1 About the shell scripts

In July, 2017, I decided to change the way that the shell scripts are designed. Users who inspect them should see the script runs R and launches a function in the `stationery` package. As a result, there should not be any difference in the result from running the script `rmd2pdf.sh` and the function inside R named `rmd2pdf`.

The only difference that might arise is when there are additional arguments supplied in the script that are not present in `rmd2pdf`.

One difference might be whether or not a table of contents is requested. The default arguments specified in the script are simple, but they do include the table of contents (this was inserted in this script in June 2017). The important line in `rmd2pdf.sh` is this: FIXME FIXME this is out of date

This turns on the table of contents feature and asks for output in the current working directory. We have not worked on command-line argument processing in `rmd2pdf.sh`, so if a user wants to change the arguments passed along, it will be necessary to edit that script file and items in the ‘defaults‘ character string.

A little lower in `rmd2pdf.sh`, one should see that the business-end of the project is a call to `Rscript` which supplies the file name, along with the ‘defaults‘ string, in a not-very-subtle way:

```
Rscript -e "library(stationery); rmd2pdf(\"$filename\", $defaults)"
```

The new arguments that users might insert in ‘defaults‘ should match the arguments that might be supplied to the `rmd2pdf` function in the `stationery` package. Hence, the help page for `rmd2pdf` should be consulted for details. Arguments that can be specified for the `render` and `pdf_document` (or `html_document` if we are creating HTML output) are spelled out. Any defaults specified in the script, or in the function call to `rmd2pdf` inside R, will override the other customizations that are place in the `stationery` package.

Some of these argument can be specified in the document header. For example, the current document header has ‘`keep_tex: false`‘. The script setting ‘keep_tex=TRUE‘ overrides that. The fact that settings can be specified several places--in the document header, or in the processing script, or in the functions that handle documents--is confusing. I don't think anybody can deny that.

Inside R, using the function ‘rmd2pdf‘, I achieve the same result by inserting the arguments into the function call:

```
rmd2html("guide-template.Rmd", toc=TRUE, output_dir="$pwd",
 clean=FALSE, quiet=FALSE, template="theme/report-boilerplate.tex",
 keep_tex=TRUE)
```

The R help page for ‘rmd2pdf‘ should be helpful (or will be, eventually).

4 Working with LaTeX

Like so many other concepts and tools in this area, TEXwas a creation of Donald Knuth at Stanford. TEXwas the precursor to LATEX.

The CRMDA maintains a Web page about LATEX: https://crmda.ku.edu/latex-help

That page has basic guides and information about the KU dissertation template. I prefer to work with LATEXdocuments, for a number of reasons. The quality of the PDF output is nicer, in my opinion, and more predictable. However, the major reason I prefer LATEXis that the Sweave option `split=TRUE` is allowed. That option creates separate `*.tex` output files, for each code chunk. The developers of the

Rmarkdown documents framework disapprove of ‘split‘ and elected, consciously, not to implement it. A couple of the questions not considered in our Web page are the following.

5 Choosing among formats

5.1 Which backend?

Should I end up with HTML or PDF? The answer depends on the intended audience/client. If a ”paper” must be submitted, obviously choose PDF. If the document needs numbered equations, cross references, and ”floating” tables and figures, choose PDF. If the document is intended for a Webpage, then HTML is the obvious choice (unless you simply want to convey a PDF document via the Web). Our HTML template includes a cascading style sheet feature that allows both color-highlighted sections and tabbed sub-sections.

There is concern about mathematics in HTML documents that deserves mention. For many years, the inability of Web pages to display equations was a major problem. Many tedious, ugly methods were developed. A more-or-less workable solution was developed, a framework called MathJax. MathJax allows inclusion of math markup in the page which--when the conditions are right--can be converted by the Web

browser to look like equations.

If the user is offline, or if the MathJax server is not available, then the HTML document’s math will not display. If one wants to put math into a document, using HTML is inherently risky. If one needs to be 100% sure that math will display as intended, choose to create PDF documents.

5.2 Which Frontend? Write in LaTeX or Rmarkdown?

This will be the answer:

> When choosing the frontend, consider the backend. Where you want to end up determines where you start.

While working on this document, I prepared an original version in Rmarkdown that was compiled into HTML. Because some features failed to compile, I changed the backend to PDF in the report style. As a result, several features that are unique to the HTML backend had to be removed. HTML offers access to some special document formatting features that are simply not available in PDF, and the converse isalso true.

I believe the following are good conclusions:

1. 1. If one intends to export as HTML, then markdown is, *by far*, the most reasonable choice for a frontend. Markdown was developed, first and foremost, as a simpler way to generate Web pages.

2. 2. If one intends to export to PDF, then markdown or LaTeX can be useful. But LaTeX is probably better. LaTeX is primarily intended for the creation of publication-quality documents in PDF format. Conversion from LaTeX to HTML is less decidedly less unsatisfactory.

Since one can put much LaTeX markup into a markdown document, perhaps the difference is not so great as it seems. The Rmarkdown compilation process (see Appendix \ref{appendix1}) generates a LaTeX file at an intermediate stage, so in some sense the same PDF result ought to be possible with Markdown or LaTeX document preparation. However, in practice, we find differences in conveniences for authors.

I would summarize the situation with a poem:

> Markdown documents intended for HTML allow some LaTeX code.
>
> Markdown documents intended for PDF allow more LaTeX code.
>
> Almost all HTML code is tolerated well in a Markdown document intended for HTML.
>
> No HTML code is tolerated in a document intended for PDF.

The main point is that if one writes a markdown document, using special features intended for the backend, then it is generally not possible to, at the last minute, change the output format from HTML to PDF, or vice versa. HTML output has advantages in Web style features, while PDF documents have advantages in ”on paper” presentations.

For novices, LaTeX seems more difficult than markdown. Perhaps this is not such a big hurdle as it used to be because [LyX](http://www.lyx.org) is available. In my opinion, ‘LyX‘ makes preparing a LaTeX document much easier than it is to prepare

a similarly complicated markdown document.

5.2.1 Important caution about Math in the HTML backend

Math is not incorporated in HTML in the same way as PDF. Compiling a document into PDF uses a program like ‘pdflatex‘ to put the equations ”in” the document. They are displayed in (more or less) the same way on various browsers and operating systems. The same is not true for math in HTML documents. Simply put, **math is not

allowed in HTML**. We think it is allowed--our eyes tells us it is

allowed--because we browse Web pages that show equations. However,

this is an illusion achieved by extraordiary measures involving

Javascript and third party servers. The beautifully formatted \LaTeX

equation is not ”embedded” in the HTML, it is instead delivered as

code ”available for rendering” in the Web browser. The HTML code is

converted, via javascript and functions supplied interactively from

the MathJax Web server.

Markdown to HTML allows most valid HTML markup, but Markdown to PDF does not allow all LaTeX.

This is a somewhat surprising difference. All HTML markup I've tried

works well in an Rmarkdown document that aims to go into HTML.

However, not all LaTeX markdown is allowed in an Rmarkdown document

going to PDF. And even less LaTeX code works well if the intended

backend is HTML. One cannot insert italics with LaTeX in a document

intended for HTML. For example, writing ‘\emph{italics}‘ or bold

‘\textbf{bold}‘ in the style of LaTeX code will have no effect in an

HTML document. However, if PDF output is used, then both of those

LaTeX codes work. As evidence, note I get \emph{italics} and bold

\textbf{bold} in this PDF document.

5.3 Should one prefer ‘Sweave‘ or ‘knitr‘?

This question is meaningful only in ‘noweb‘/ LaTeX documents. In Rmarkdown, ‘knitr‘ is the only available method to process code chunks. In ‘noweb‘, one can choose between ‘Sweave‘ and ‘knitr‘. Perhaps that suggests that, if one must learn one set of chunk options, then ‘knitr‘ options are the right place to start (since they can be used in documents intended for HTML or PDF). The chunk options allowed the original ‘Sweave‘ were ‘echo‘ (include code with output?), ‘eval‘ (run the chunk calculations?), ‘include‘ (display the chunk in the document?), ‘fig‘ (code generates a figure?) and ‘results‘ (output in LaTeX is handled differently than raw \TeX). There are a few others, but that is most of the story. ‘knitr‘ honored most of the Sweave options and then added many more (see [knitr code chunk options](https://yihui.name/knitr/options)).

One benefit of Rmarkdown with ‘knitr‘ is that it is possible to make documents about other programs (not just R). I've explored knitr to weave documents about BASH shell programming, for example.

6 Troubleshooting

The first step is understanding the trouble. The trouble stems from the fact that each document must be transformed through several stages to reach the final result. Understanding that, and learning about the problems that appear at each stage, can help with the troubleshooting strategies that we recommend.

6.1 Compilation Stages\label{appendix1}

Steps to compile documents break down into 2 phases.

1. Handle code chunks

2. render the resulting document.

A noweb file is converted from Rnw into PDF by a sequence of transitions.

1. ‘Rnw -> tex‘. This is called "weaving" or "knitting", depending on whether Sweave or knitr is the code processing engine. R finds the code in the Rnw file and inserts results into a new LaTeX file. The difference between weaving and knitting will be explained below.

2. ‘tex -> pdf‘. The default is ‘pdflatex‘ for this step, but the alternative ‘xelatex‘ is growing in popularity because it more gracefully handles Unicode characters (utf8). If the document is edited with [LyX](http://www.lyx.org), there is an

implicit step 0,

0. ‘LyX -> Rnw‘.

In the LyX pull down menu system, this is represented by Export -> Sweave.

A markdown file is converted from Rmd to HTML by this sequence of transitions

1. ‘Rmd -> md‘. The "knitting" process replaces code chunks by R input and output, converting the R markdown file into an ordinary markdown file. In my system, an "md" file is generated, and then a second "utf8.md" is generated to clean up the file encoding.

2. ‘md -> HTML‘. Currently, most people use the program ‘pandoc‘ for this. A version of ‘pandoc‘ is distributed with Rstudio for the convenience of users. Linux users probably have ‘pandoc‘ available as standard system packages and the Rstudio version be removed.

The production of PDF from markdown, involves an additional transition.

1. ‘Rmd -> md‘. Knitting converts code chunks into R input and output that is inserted into an ‘md‘ file created in the ‘pandoc‘ markdown style.

2. ‘md -> tex‘. A LaTeX file is created by ‘pandoc‘. In the header of the ‘md‘ document, one can set a number of parameters to alter the LaTeX generation process. For troubleshooting, ”keep_tex: yes” to keep the ‘tex‘ file.

3. ‘tex -> pdf‘. The default program for this has been ‘pdflatex‘. It may be important to know that a LaTeX document may need to be run through ‘pdflatex‘ several times because cross-references among pages and equations need to be made consistent.

The R packages ‘rmarkdown‘ and ‘knitr‘ orchestrate the process that builds the instructions to ‘pandoc‘. In ‘rmarkdown‘, the function ‘render‘ orchestrates all of the work. It calls chunk calculator and assigns work among the various conversion programs. The functions in the stationery package named ‘rmd2pdf‘ or ‘rmd2html‘ are ”wrapper” functions that adjust settings sent to ‘render‘.

6.2 Avoiding compilation trouble

The document production phase can fail at many steps (see Appendix \ref{appendix1}). While editing a document, authors are well advised to heed the advice:

\begin{quote}

\textbf{Compile early, compile often!}

\end{quote}

When a mistake is inserted, it is best to find it as soon as possible.

6.3 When debugging, check intermediate files

The compiler scripts may erase intermediate files. While debugging a document, we want to disable that clean-up step so that we can see what goes wrong. In an Rmd document that ends up in PDF, for example, we should be able to inspect an ‘md‘ file and a ‘tex‘ file. We can not only inspect those files, but we can also attempt to compile them in isolation so that we can see what is going wrong.

This document includes the header argument ‘keep_tex: true‘, which means we save a copy of ”reports_and_guides.tex”.

While developing this document, some of the problems with backend-specific code have come to the forefront. The HTML backend allows pleasant color-coded section headings which included in the PDF output. Tables that work well in PDF documents don’t work in HTML, and vice versa. Some HTML tables that are legal HTMLdon’t cooperate with ‘pandoc‘.

It is also worth mentioning that error messages are not always informative. In fact, we sometimes don’t get error messages when we should. Rather, we simply receive bad output. While developing this document I noticed that when a user includes erroneous LaTeX code in a markdown document, a flawed HTML output is generated without error or warning. On the other hand, changing the intended backend to PDF causes the compiler to fail and issue an error message. If one is exporting to HTML, then, a very careful proofreading of the output to check conversion of LaTeX code into HTML is necessary.

7 What do we Really, Really Need?

Things we wish we could have in HTML output (that we can get in PDF output)

Numbered equations, easy cross references, numbered tables and figures

Things we wish we could have in PDF output (that we can get in HTML)

A splash of color, mainly. This is possible in PDF, but more

difficult, at least on the surface.

7.1 Math

We are a Center focused on methodology. It is necessary to be able to

write about math, preferably with a standard, uniform mathematical

markup language, such as \LaTeX.

Many social scientists are not familiar with LaTeX document

preparation. That was a hurdle that kept many authors with Microsoft

Word, even when they were frustrated with it. The difficulty of using

LaTeX was solved, to a significant extent, by LyX, an open source

graphical interface. With LyX, or other editors that could generate

LaTeX output (such as Scientific Word, TexMacs, or Abiword), authors

who were not computer programmers could learn enough LaTeX to finish

their projects.

7.2 Literate documents: include code and output

We need to be able to write about computer code. The "old fashioned"

way is to copy/paste code and output into documents. That's somewhat

error prone and difficult to keep up-to-date.

Donald Knuth, a famous Stanford professor of computer science, proposed strategies to integrate the production of documents with the development of computer code. Rather than creating code in one file, and documentation in another, the idea was that the two parts of our work should be blended in a "literate programming" exercise.

The literate programming idea is more a general way of life than it is a particular document production strategy. It is, partly, aimed at programmers who don't like to write instruction manuals. In the end, however, it may have more impact on non-programmers who need to prepare technical reports that include computer code examples.

In computer programming, one of the biggest impacts of literate programming is the proliferation of systems for preparation of documentation within code files. In the 1990s and early 2000s, when I was working on the Swarm Simulation System, we used a framework called Autodoc that allowed us to write instructions into Objective-C code

that were later harvested and turned into instructional manuals. See, for example, the documentation for the [Opinion Formation model]

(http://pj.freefaculty.org/Swarm/MySwarmCode/OpinionFormation/Opinion-Docs). The

Autodoc program was poorly documented and not easy to get, but soon after that, a new coding system called Doxygen became widely available. Doxygen, developed for creation of instruction manuals for C++ programs, was a major success in computer programming. Like Autodoc, Doxygen gave programmers a relatively convenient method to explain what they were doing without wasting too much time.

In the modern experience of most CRMDA staffers, the "documentation inside code" approach is visible in the Roxygen markup method used for functions in R documents. Any function worth using should have Roxygen markup.

7.3 Where we have been

In order to embrace the importance of using either markdown or \LaTeX, one must first abandon the idea that Microsoft Word can ever be useful for serious authorship. That's a big step for many graduate students and professors.

If we look past GUI "what you see is what you get" (*wyswig*) word processors, where do we go? For a long time, the only answer was \LaTeX. However, there was a fatal weakness in \LaTeX. It is intended for PDF output, not Web pages. Exports into HTML were problematic. LaTeX was not only difficult for some to use, but it also did not benefit from fancy features that were becomming available in the Internet, especially cascading style sheets and Javascript.

This gap in the document production process created a need for a new methods. In the 1990s, there was quite a bit of effort to make user friendly Web page editors, so that authors could have a Word-like experience that would generate HTML. The end result, generally, was difficult-to-maintain HTML documents. It was generally not feasible to edit and revise documents, the HTML generated was both extremly complicated and generally unsatisfactory.

Another strategy was the development of alternative markup languages. In a way similar to \LaTeX, these markup languages (e.g., 'docbook') drop the idea that the user should have a *wyswig* experience. Instead, the author would again become a programmer who would insert symbols to create sections.

Markdown was developed as a rejection of both ugly markup documents (either LaTeX or HTML) and *wyswig* editors. The idea is that documents should be text files that are readable *as is* but also convertible into other backends. The "markdown" movement seeks to deliver an easier-to-edit, less difficult-to-read, and easier-to-convert format. The leader is John Gruber, whose Website is boldly named ["daringfireball"](https://daringfireball.net/projects/markdown).

Markdown is intended to be easy-to-read, so that even if it is not compiled into a backend, it might be presentable to an audience of non-programmers. That general idea seems unrealistic to me, there is almost never going to be an audience (for CRMDA, at least), which is eager to look at a markdown file. The dropback argument is that a markdown file is more easily produced by a novice who does not want to learn to use \LaTeX. This seems more persuasive.

The strength of markdown is that it makes it fairly easy to produce HTML documents that utilize some (not all) of the strengths of the World Wide Web's most commonly used method of communication.

Where markdown is not capable, one can still write ”raw HTML” in the middle of a markdown document.

8 Required footer

The guide documents we require authors include a final chunk that includes the session information, to be used in bug-tracking.

Reports do not include raw output, so it is not recommended to insert that raw output in the report. Instead, we ask report writers toinclude a final R chunk that saves the session information in a file in the same directory as the pdf output.

‘‘‘{r session, include=F}

zz <- ”stationery-sessionInfo.Rout”

capture.output(warnings(), file = zz)

capture.output(sessionInfo(), file = zz, append = TRUE)

‘‘‘

Available under

[Created Commons license 3.0 <img src=”http://crmda.dept.ku.edu/images/cc-3.0.png” alt=”CC BY”

style=”width: 75px;height: 20px;”/>](http://creativecommons.org/licenses/by/3.0/)

References

References

(2013). Implementing reproducible research. *Ebook library.*

Knuth, D. E. (1984). Literate programming. *The Computer Journal. British Computer Society*, 27(2), 97–111.

Leisch, F. (2002). Dynamic generation of statistical reports using literate data analysis. In W. Härdle & B. Rönz (Eds.), *Proceedings in Computational Statistics*. Physika Verlag.

R Core Team (2017). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria.

Sewell, W. (1989). *Weaving a Program: Literate Programming in Web*. Van Nostrand Reinhold Computer.

Xie, Y. (2014). knitr: A comprehensive tool for reproducible research in R. In V. Stodden, F. Leisch, & R. D. Peng (Eds.), *Implementing Reproducible Computational Research*. Chapman and Hall/CRC. ISBN 978-1466561595.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Boca Raton, Florida: Chapman and Hall/CRC, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. Chapman and Hall/CRC.

Xie, Y. (2018). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.19.

```
zz <- "stationery.Rout"
capture.output(sessionInfo(), file = zz, append = FALSE)
if (!is.null(warnings())){
     capture.output(warnings(), file = zz, append =  TRUE)
 }
```