The OdfWeave Package

Max Kuhn
max.kuhn@pfizer.com

September 5, 2006

1 Introduction

The `Sweave` function (Leisch, 2002) is a powerful component of R. It can be used to combine R code with LaTeX so that the output of the code is embedded in the processed document. The capabilities of `Sweave` were later extended to HTML format in the `R2HTML` package.

A written record of an analysis can be created using `Sweave`, but additional annotation of the results may be needed such as context–specific interpretation of the results. `Sweave` can be used to automatically create reports, but it can be difficult for researchers to add their subject–specific insight to pdf or HTML files.

The `odfWeave` package was created so that the functionality of `Sweave` can used to generate documents that the end–user can easily edit.

The markup language used is the Open Document Format (ODF), which is an open, non–proprietary format that encompasses text documents, presentations and spreadsheets. Version 1.0 of the specification was finalized in May of 2005 (OASIS, 2005). One year later, the format was approved for release as an ISO and IEC International Standard.

There are several editors/office suites that can produce ODF files. OpenOffice is a free, open source editor that, as of version 2.0, uses ODF as the default format. `odfWeave` has been tested with OpenOffice to produce text documents. As of the current version, `odfWeave` processing of presentations and spreadsheets should be considered to be experimental (but should be supported in subsequent versions). OpenOffice can be used to export the document to MS Word, rich text format, HTML, plain text or pdf formats.

One advantage to using `Sweave` with ODF files is that no experience with markup languages is needed, so a broader set of users can create documents. However, the Open Document Format is very new and may go through significant changes in the future

Users can create documents with `Sweave` commands in ODF using (almost) exactly the same

format as required when using LaTeX markup. A basic call to `odfWeave` looks like

```
odfWeave(inFile, outFile, workDir = odfTmpDir(), control = odfWeaveControl())
```

where `inFile` and `outFile` are the source and destination file names, `workDir` is a path where the files are processed and `control` is a control object that can be used to specify image formats and style specifications. The functionality of `odfWeave` is described in more detail later.

2 Requirements

To use `odfWeave`, the user must have a basic understanding of `Sweave`.

`odfWeave` requires files to be in the Open Document Format, version 1.0 or above. These can be generated by OpenOffice version 2.0 or above (see Section 3). The package also requires a utility to zip and unzip compressed files, such as `unzip`[1], `Winzip` or `jar`.

Also, by default, `odfWeave` tries to save images in png format. In Unix and Linux, a png device may not be available. There are three options if this is the case: enable the png device, using the `bitmap` device (which requires GhostScript) or specify an alternate image format.

3 The Open Document Format

The Open Document Format is a document format that encompasses text documents, spreadsheets, presentations and other types of files. The document extension depends on the document type: odt for text documents, odp for presentations and so on. Open Document Format files are compressed archives of several files and folders which can be decompressed using standard tools such as `unzip`, `jar` or `WinZip`. Some resources for the format are:

- The format specification (OASIS, 2005)
- "Introduction to the format internals" by David Carrera
- "OASIS OpenDocument Essentials – Using OASIS OpenDocument XML" by J. David Eisenberg

A typical document will have a structure similar to:

[1] a free utility available for many operating systems at http://www.info-zip.org/

```
Name
----
content.xml
layout-cache
META-INF/
META-INF/manifest.xml
meta.xml
mimetype
Pictures/
Pictures/rplot.png
settings.xml
styles.xml
Thumbnails/
Thumbnails/thumbnail.png
```

There are sub–directories in the archive:

- `META-INF/` contains manifest.xml, which enumerates the entries in the compressed archive
- `Pictures/` contains any image files that are included in the document.
- `Thumbnails` has images of the rendered document.

Additionally, the files contained in the compressed archive include:

- `content.xml` contains the content of the document (e.g. text paragraphs, tables, etc.)and some formatting.
- `meta.xml` contains summary information about the document, such as creation date, number of edits, document statistics (e.g. number of words, etc) and the identification of the application that generated the document.
- `settings.xml` lists the configuration of the document, such as the zoom when opened or printing options.
- `styles.xml` has formatting information for almost all of the elements in the document, such as fonts or table formatting.

Since ODF is based on XML, the underlying markup tends to be very verbose. For example, the in–line `Sweave` expression:

```
\Sexpr{paste(letters[1:5], collapse = ",")}
```

produces the following markup in `content.xml`:

```
<text:p text:style-name="SomeStyleDef">
\Sexpr{paste(letters[1:5], <text:s text:c="2"/>collapse = &quot;,&quot;)}
</text:p>
```

Code chunks yield similar formatting:

```
<<figureTest2,fig = TRUE,echo=FALSE,results=hide>>=
library(lattice)
out <- densityplot( randomData, adjust = 1.5)
print(out)
```

The resulting markup in `content.xml` would look like:

```
<text:p>&lt;&lt;figureTest2,fig = TRUE,echo=FALSE,results=hide&gt;&gt;=</text:p>
<text:p>library(lattice)</text:p>
<text:p>out &lt;- densityplot( randomData, adjust = 1.5)</text:p>
<text:p>print(out)</text:p>
<text:p>@</text:p>
```

A significant portion of the `odfWeave` code base is for pre–processing the XML files so that they can be passed to the `Sweave` function in conjunction with a custom ODF driver. All R code written in ODF documents will be encased in XML tags, which isn't an issue for in–line commands, but these must be stripped for code chunks. Also, several characters (">", "<", "&", single quotes and double quotes) are automatically converted to alternative representations ("`&gt`", "`&lt`", "`&amp`", "`&apos`" and "`&quot`"). Also, two or more consecutive spaces are represented using XML tags. One additional complication arises from the automatic character conversion features of some editors. For example, minus signs ("-") are sometimes converted to long dashes ("–"). These characters must be caught and converted before they are sent to the R parser.

Given the structure of an ODF file, `odfWeave` must decompress the file, pre–process the XML files, Sweave and then re–compress the archive.

4 Using odfWeave

The functionality of `Sweave` is mostly preserved in `odfWeave`, such as weaving, hooks, figure environments, etc. Some functionality, such as writing output to separate files for each code chunk using

the `split` argument, doesn't make sense when using ODF. As another example, ODF supports a broad range of image formats, so `pdf` or `eps` arguments to code chunks are somewhat limiting.

In–line `Sweave` expressions are created using `\Sexpr`. Code chunks can be created with the noweb convention (using `<<>>=`). The LaTeX code chunk syntax is not currently supported.

The image format and sizes are specified using `getImageDefs` and `setImageDefs`. For example:

```
> getImageDefs()
$type
[1] "png"

$device
[1] "png"

$plotHeight
[1] 480

$plotWidth
[1] 480

$dispHeight
[1] 5

$dispWidth
[1] 5

$args
list()
```

This shows that the type of image to generate is "png" via the `png` device driver. The `plotHeight` and `plotWidth` values are set in the units of the particular device driver (pixels in this case, but other devices use inches). The dimensions of the image file and the dimensions of the rendered image can be set independently using `dispHeight` and `dispWidth` (always in inches). The `args` element is a list of graphics device arguments. For example, when using postscript graphics, you might need to use the options `horizontal = FALSE`, `onefile = FALSE` and `paper = "special"`.

To change the defaults, this list can be saved and edited. The modified version can be used via `setImageDefs(modObject)`, where `modObject` is a modified version of the above list. This can be done prior to calling `odfWeave` or can be set in–between figure code chunks.

The `odfTable` class can be used to convert vectors, matrices and data frames to native ODF tables, much like the `latex` function in the `Hmisc` package. The `odfCat` can be used to write out text in native ODF format. There are also functions to created bulleted lists and inserting external

images (i.e. images not created in a chunk with `fig = TRUE`).

Figure 1 shows an example text document prior to weaving. For illustrative purposes, we might want to adjust the display size of the image to a 4.5 inch square. The R code to do this is:

```
library(odfWeave)
inFile <- "example.odt"
outFile <- "out.odt"

imageDefs <- getImageDefs()
imageDefs$dispWidth <- 4.5
imageDefs$dispHeight<- 4.5
setImageDefs(imageDefs)

odfWeave(inFile, outFile)
```

Figure 2 shows the results. Note that since the first in–line `Sexpr` command had uniform formatting, the output of that command retained the formatting. In general, the formatting of in–line `Sweave` tags and code chunks is required to be uniform (as an error is likely to be produced otherwise). For example, `round(`*sqrt(2)*`, 2)`, ends up looking like

```
<text:p>round(<text:span text:style-name="T10">sqrt(2)</text:span>, 2)</text:p>
```

and can be difficult to process and parse.

One additional note about this example: the initial code chunk that loads the data does not specify the full path to the file. In this example, the source data were previously zipped into the ODF file (using `zip example.odt pcrData.csv`) so that the source data are embedded with the analysis and report. Once the file is processed using `odfWeave`, the resulting document also contains the source data in its original form.

5 Formatting

There are two main components to specifying output formats: style definitions and style assignments. The definition has the specific components (such as a table cell) and their format values (e.g. boxed with solid black lines). The function `getStyleDefs` can fetch the pre–existing styles in the package. For example:

```
> getStyleDefs()$ArialNormal

$type
[1] "Paragraph"
```

```
<<loadData, results = hide, echo = FALSE>>=
# Polymerase chain reaction (PCR) data for 3 dose groups
pcrData <- read.csv("pcrData.csv")
@
```

There were **\Sexpr{dim(pcrData)[1]}** subjects measured across \Sexpr{length(unique(pcrData$Compound))} drug groups. A density plot of the data is produced with the lattice package:

```
<<densityPlot, echo = FALSE, fig = TRUE>>=
library(lattice)
trellis.par.set(col.whitebg())
print(
   densityplot(
      ~log(Cycles, base = 2),
      pcrData,
      groups = Compound,
      adjust = 1.5,
      pch = "|",
      auto.key = list(columns = 3)))
@
```

Here is a table of the mean cycles to threshold for each drug group:

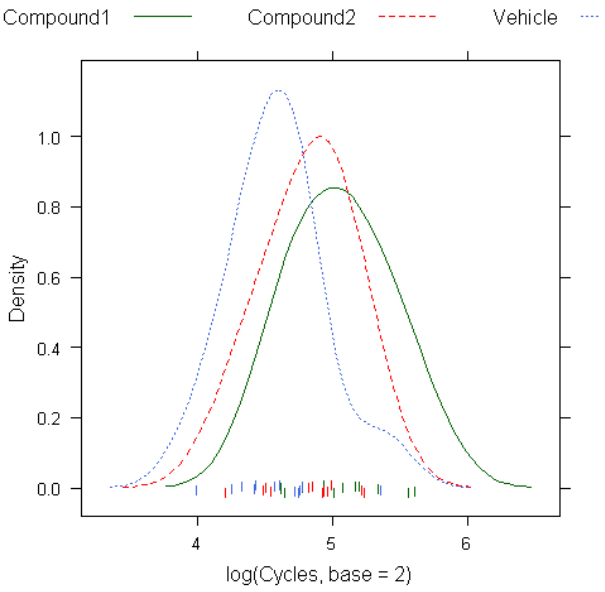
```
<<meanTable, echo = FALSE, results = xml>>=
meanCycles <- tapply(
   log(pcrData$Cycles, base = 2),
   pcrData$Compound,
   mean)

odfTable(
   meanCycles,
   horizontal = TRUE)
@
```

Of course, we would normally look at diagnostics before going straight to the p-value

```
<<lmFit, results = verbatim>>=
linearModel <- lm(
   log(Cycles, base = 2) ~ Compound,
   data = pcrData)
anova(linearModel)
@
```

Figure 1: An example of an ODF document containing `Sweave` tags.

There were **36** subjects measured across 3 drug groups. A density plot of the data is produced with the lattice package:



Here is a table of the mean cycles to threshold for each drug group:

Compound1	Compound2	Vehicle
5.054	4.816	4.590

Of course, we would normally look at diagnostics before going straight to the p-value

```
> linearModel <- lm(log(Cycles, base = 2) ~ Compound, data = pcrData)
> anova(linearModel)
Analysis of Variance Table

Response: log(Cycles, base = 2)
          Df Sum Sq Mean Sq F value   Pr(>F)
Compound   2 1.2947  0.6473   5.829 0.006794 **
Residuals 33 3.6648  0.1111
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 2: The processed ODF document.

```
$parentStyleName
[1] ""

$textAlign
[1] "left"

$fontName
[1] "Arial"

$fontSize
[1] "12pt"

$fontType
[1] "normal"

$fontColor
[1] "#000000"
```

These can be modified and new definitions can be added. The function `setStyledefs` "registers" the style changes with the package. When `odfWeave` is called, these definitions are written to the style sections of the XML files. There is a second mechanism to assign styles to specific output elements. The functions `getStyles` and `setStyles` can be used to tell odfWeave which style definition to use for a particular output:

```
> currentStyles <- getStyles()


$paragraph
[1] "ArialNormal"

$input
[1] "ttRed"

$output
[1] "ttBlue"

$table
[1] "Rtable1"

$cell
[1] "noBorder"
```

```
$header
[1] "lowerBorder"

$cellText
[1] "ArialCentered"

$headerText
[1] "ArialCenteredBold"

$bullet
[1] "Rbullet"
```

For example, the `input` and `output` elements control how R code and command–line output look. To change either of these, an existing definition can be assigned to these entries and reset using `setStyles(currentStyles)`. Unlike the style definitions, the style assignments can be modified throughout the R code.

The package also contains a function, `tableStyles`, that can be used to differentially format specific cells and text in tables.

6 Other Functions

There are a few miscellaneous functions also included in the package. `pkgVersions` can create a summary of the packages and versions in the search path:

```
> pkgVersions("matrix", ncol = 3)
```

```
     [,1]                  [,2]                  [,3]
[1,] "base (v.2.3.1)"      "lattice (v.0.13-8)" "tools (v.2.3.1)"
[2,] "datasets (v.2.3.1)"  "methods (v.2.3.1)"  "utils (v.2.3.1)"
[3,] "grDevices (v.2.3.1)" "odfWeave (v.0.4.4)" ""
[4,] "graphics (v.2.3.1)"  "stats (v.2.3.1)"    ""
```

The function `listString` takes a vector and returns a textual list. For example, `letters[1:4]` would become ”a, b, c and d”. Also, `matrixPaste` can take a series of character matrices with the same dimensions and perform an element–wise paste.

7 Converting ODF to Other Formats

Using OpenOffice, ODF files can be manually converted to other formats using the "Save As" or "Export" items in the File menu. To convert documents using a command line interface, there are at last two options:

- on platforms with the Bash shell, Nathan Coulter has written a script that uses the Python internals that are installed with OpenOffice, called ooconvert. This can be found at http://sourceforge.net/projects/ooconvert.
- there is a Java class, called JOOConverter, that is available at http://jooreports.sourceforge.net/ that can also convert documents.

.

8 References

Carrera, D. (2005). *Introduction to the format internals.* URL: http://opendocumentfellowship.org/Articles/IntroductionToTheFormatInternals.

Eisenberg, J. D. (2005). *OASIS OpenDocument Essentials – Using OASIS OpenDocument XML.* URL: http://books.evc-cit.info/.

Leisch, F. (2002). *Sweave, Part I: Mixing R and LaTeX.* R News, 2(3):28–31. URL: http://cran.r-project.org/doc/Rnews.

Organization for the Advancement of Structured Information Standards (OASIS) (2005). *Open Document Format for Office Applications (OpenDocument)* v1.0. URL: http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf