Writing Vignettes for Bioconductor Packages

R. Gentleman

January 30, 2014

This document details some of the important concepts that should be used and adhered to when writing *vignettes* for Bioconductor packages.

A vignette is a navigable document with dynamic content. Sweave (**?**) provides one system for producing and working with these documents. A number of tools for dealing with vignettes are provided by the R package *tools* while others are contained in the Bioconductor package *DynDoc*. Another source of information regarding vignettes and their treatment during package building and checking is provided by the R Extensions manual.

Some Macros

In writing LaTeX we recommend using the macros `\texttt{foo}`, `\textbf{foo}` and `\textit{foo}` in preference to the older style of `{\em foo}`. An important reason for preferring the former is that it is easier to do search and replace on them.

It is also recommended that you make use of specific macros for identifying R functions, objects and macros. The more popular ones are listed below.

- **Robject** `\newcommand{\Robject}[1]{{\texttt{#1}}}`
- **Rfunction** `\newcommand{\Rfunction}[1]{{\texttt{#1}}}`
- **Rpackage** `\newcommand{\Rpackage}[1]{{\textit{#1}}}`
- **Rclass** `\newcommand{\Rclass}[1]{{\textit{#1}}}`
- **Rmethod** `\newcommand{\Rmethod}[1]{{\textit{#1}}}`
- **Rfunarg** `\newcommand{\Rfunarg}[1]{{\textit{#1}}}`

Putting these in the header of the `.Rnw` file makes them available in LaTeX.

In addition there are some issues regarding some control statements for vignettes. These are typically included as LaTeXcomments at the very top of the vignette. For example:

```
%\VignetteIndexEntry{Annotation Overview}
%\VignetteDepends{Biobase, genefilter, annotate}
%\VignetteKeywords{Expression Analysis, Annotation}
%\VignettePackage{annotate}
```

These different identifiers are used as follows:

- **VignetteIndexEntry**: will be used as the entry in the file `/inst/doc/00Index.dcf` that is created when the package is built. This will then be used by functions such as `openVignette` and will be used on menus to identify the vignette. So please use short, unique, meaningful names here.
- **VignetteDepends**: is a comma separated list of the packages that the vignette depends on. At some point tools for automatically collecting and installing these packages will be developed.
- **VignetteKeywords**: this is a comma separated list of keywords and phrases. At some point we hope to provide an ontology to select from and some tools for locating vignettes according to specified keywords. For now this is mainly informational.
- **VignettePackage**: this indicates the name of the package that the vignette *belongs* to.

In some cases the name of the file (minus the `.Rnw` suffix) is used to identify the vignette. To minimize confusion it will be easiest if the name of the file and the `VignetteIndexEntry` are the same.

Some Words Of Caution

Please remember that the vignettes get processed when the package is built. Any functionality that requires interactivity, such as opening a web page, opening a widget etc. must be placed inside a conditional statement that is only evaluated when R is interactive.

For example,

```
if(interactive() )
  vExplorer()
```

So that R only tries to open the `vExplorer` widget when there is a user to interact with it.

Please also consider whether you want all R commands to be output. In some settings (particularly developing laboratory materials there are good arguments for explicitly exhibiting every command but in other situations some materials should be hidden. For some descriptions it is more important to explain the important commands and to hide the details that get the reader to them. Remember, they can always reconstruct the whole sequence using `vExplorer` and some of the other tools that have been made available.

Note also that `Sweave` is not very smart; it is basically just doing text substitution and so it does not know whether your R code is in a verbatim environment or even located outside of the `begindocument`, `enddocument` control statements. `Sweave` simply searches for the correct syntax (either a construct like `<<>>=` in position one on a line or `@` in position one on a line and interprets it accordingly.

Please use informative labels on the code chunks. This helps others understand what is going on and the names of the chunks are about all that the users see when using `vExplorer`. The current limitations on text output to `vExplorer` also suggest that you might want to include a few comments in the code chunk to explain, briefly, what is going on.

Some Tools

In *tools* the basic functions for weaving (processing the document to get dvi or pdf output) and for tangling (extracting the code chunks to a file) are provided. As part of *DynDoc* we provide a function called `tangleToR` that extracts the code into an R object that can subsequently be used by the reader to evaluate the different code chunks.

The functionality provided by `vExplorer` will perhaps be the most use to new users as it allows them to step through the code chunks sequentially. We will be working on substantial improvements to this interface for the 1.3 release of Bioconductor.

Helping Users Find And Use Your Vignette

Perhaps the hardest task that we face is ensuring that vignettes are easily available to the users of the system. One way to do that on Windows (we could do something similar for the Gnome interface), is to include the following code snippet:

```
if(.Platform$OS.type == "windows" && require(Biobase) && interactive()
    && .Platform$GUI ==  "Rgui"){
    addPDF2Vig("pkgName")
}
```

where `pkgName` is the name of a package. This code checks to see if it makes sense to try and add a new menu item `pkgName` to the menu `Vignettes` and if so does just that. For example, `addPDF2Vig("widgetTools")` will add a new mune item `widgetTools` to the menu `Vignettes`, which has clickable names for all the vignettes contained by *widgetTools*. Currently, `addPDF2Vig` is in *Biobase*, meaning that *Biobase* is required for adding vignettes to a menu. Efforts are being made to remove the dependency on *Biobase*.

Answers To Common Questions And Issues

The rest of this document will be devoted to answering common questions and issues that arise when users deal with vignettes.

My Vignette Does Not Build Properly

If a user is having a problem with their vignette not building properly, there can be several issues involved depending on how they are building the vignette. It is important to know the various mechanisms by which these files are built. Depending on the situation, one might be calling `Sweave`, `buildVignettes` or `checkVignettes` (the latter two call `Sweave`, however). The `Sweave` function should report which code chunk caused the error and what that error was (although it might have been due to a bug in a previous code chunk).

If the problem is arising during the operations `R CMD check` or `R CMD build`, then the functions being used are `checkVignettes` and `buildVignettes` respectively. It is important to note that these are run in a particular computational environment - simply starting up R and calling these functions will not necessarily produce the same results. It is important to start R with the options *–vanilla* and *–quiet* when loading R to start either of these functions and *–no-save* for `buildVignettes` in order to properly recreate the environment used.

In all cases the primary task for the user here is to find out what is wrong with the actual code and solve that problem. One method that can be used is trying the function `Stangle` or `vExplorer` to go through each code chunk interactively within R.

My Vignette Builds Too Slowly

This can happen sometimes when the code chunks in the document are themselves very computationally intensive. Since the code chunks are run when the vignette gets built, a vignette can end up taking an extremely long time to generate. If a user finds themselves in this situation, there are a few easy steps that can be taken to alleviate the problem.

One solution to this problem is to split the file up into multiple vignettes, each describing a smaller piece of the overall puzzle. This might not be feasible for various reasons - the vignette might have already been dealing with an atomic topic for instance. Also, if the user is creating an R package, putting multiple vignettes in the package will take just as long as having one very large package when building the package.

Another potential solution is to try and use precomputed bits of code wherever possible. These could be stored as data files in a package or some other similar location and then loaded instantly instead of having them be actually computed in the code chunks. Or, similarly, the author might be able to construct simpler examples (which might very well be less useful in a 'real world' sense).