Getting started with rtemplate

Sebastian Weber, Inga Ludwig

2016-12-12

Intro

This is a template for the package's main vignette. It also outlines the main components required for the validation of Novartis R packages and provides some tips & tricks.

Note: A pdf version of this vignette also serves for documentation of the CSV process for scientific software developed in R. To reproduce, checkout the `rtemplate` code on `http://chbslx0132.eu.novartis.net/svn/rtemplate` and build the vignette document in pdf format (note: default for R packages is html, so the output format `html_vignette` must be replaced by `pdf_document` for this purpose).

The `rtemplate` has been developed based on the `RBesT` package experience, for which a new Computer System Validation (CSV) model for R packages was established in collaboration with DevIT and QA. It is supposed to ease development and validation of scientific software written in R at Novartis.

Key of the validation approach is to avoid duplication of work by mapping the elements of best practice in software development (specification, documentation, testing) with the quality requirements of the validation process. The following key R package deliverables are required to ease the subsequent validation steps and facilitate a standardized deployment process on Modesim:

R package deliverable	R implementation	Novartis validation
Vignette	`vignettes/introduction.Rmd`	User specifications (USR)
R reference manual	Roxygen2 comments	Functional specifications (FS)
Unit tests	`tests/testthat/test-*.R`	User acceptance tests (UAT)

Each item will automatically be formatted correctly during the installation process of the package and result in individual files. During the deployment process on MODESIM these files are transferred into the PROTON system which is the software system of QA to record test evidence. Therefore, all software relevant documentation will be automatically generated from the R package source and will be automatically updated with each release of the R package to MODESIM.

In the following, you'll find a more detailed instruction on how to use this template to implement your own Novartis standard compliant R package. For details on the coding standards to be followed when implementing re-usable R code, see the Novartis guideline on Scientific Software in R [1] and references given therein. For a more general introduction into R package development and best practices, see Hadley Wickham's books on R packages [2] and Advanced R [3].

Following best practices in software development may seem to create additional work in comparison to "just write code". However, it has substantial value, given the R package will be self-documented, tested and hence future maintenance will be ensured. Finally, following these principles makes the Novartis CSV process for new R packages extremely lean and reduces to minor adjustments of the validation documents developed for `RBesT`.

User specifications: the vignette

As a minimum requirement, your package must contain at least one vignette called `introduction`, which serves as a starting point to describe the functionality of your package (you may add further vignettes as you

wish). Make sure to describe the purpose of the software and to give full description of its most important functionality. You should also provide examples that will be executed when your vignette is built.

The vignette will serve as a representation of the user requirements for your package. This imposes certain requirements on how the functionality is described: Describe the functionality in steps that reflect the major steps of the workflow that your package is supposed to support. Note that this step structure must be reflected in the way you organize the unit tests of your package (see below).

The following is a dummy example of how a vignette might be structured for the example functionality that is included in this `rtemplate` (see the implementation of the corresponding unit tests to see how both are aligned):

How a vignette for the `rpackage` example functionality might look like:

The `rtemplate` package has been created to facilitate easy implementation, validation and deployment of analysis code written in R in accordance with the Novartis QA requirements. The `rtemplate` features also a couple of examples the illustrate the implementation of a dummy workflow.

1. Execution of basic arithmetic operations

```
library("rtemplate")

# Calculate the sum of 1 and 2
sum12 <- add(1, 2)
sum12
```

[1] 3

```
# Do also some subtraction
subtract(2, 1)
```

[1] 1

2. Retrieve some dummy data: The result can be used to lookup some data items that are provided by this package, using the `retrieve_data` functionality:

```
# Retrieve the dataset 1 data item at position sum12
retrieve_data(sum12)
```

[1] 3

```
# Retrieve the dataset 2 data item at position sum12
retrieve_data(sum12, dataset=2)
```

[1] 30

Done. To wrap up, your vignette should also contain a list of references (if applicable) and the R `sessionInfo` output (see below). For a more realistic example, see the vignette of the `RBesT` package (it is installed in Modesim - just do `library(RBesT)`).

Functional specifications: the R reference

Make sure to document all objects and functions in your code, using `roxygen2` notation like in the included examples. Describe all parameters, default settings and expected behavior (don't forget the handling of exceptions) and add examples. Adapt `rtemplate-package.R` to contain the top-level help information on your package, including contact information and the version history.

The reference PDF manual is created automatically whenever the R package is built. The PDF will be stored in `inst/doc` and is available to the user via the R help system.

User acceptance tests: unit testing

All functionality of your package should come with unit tests (see the included examples on how to set up unit tests with `testthat`). Note that the structure of the tests must reflect the order of the steps in which your vignette describes the workflow that is supported by your package: one `test-*.R` file per step, with the order in which the files are called in `run-oq.R` matching the order of the descriptions in the vignette. Tests of any additional functionality that is included in the package, but not described in the vignette goes to `test-misc.R`, which is executed last.

An objective metric which we highly recommend to periodically check is the test coverage. This quantity is the percent number obtained by dividing the number of lines of code which get executed whenever all tests of the package are run by the total number of lines of code. To determine the test covarge of your R package you may use the `covr` package like:

```
library(rtemplate)
library(covr)

# Change into the root directory of your package first!

pkg_cov <- package_coverage()

# Percent coverage is STRONGLY RECOMMENDED to be at least 90%
percent_coverage(pkg_cov)
# More details:
print(pkg_cov)

# You can also easily explore which code parts are not yet tested well
# (this will call the shiny package)
shine(pkg_cov)
```

For your ease, there is a script called `coverage.R` included in the `utils` folder, that you can run from the base directory of your package to perform the above described steps.

Creating a new project from the `rtemplate`

Step 1: Get the package template Make sure you have obtained the R package template from SVN via the command

```
svn export http://chbslx0132.eu.novartis.net/svn/rtemplate <your_package_name>
```

Note:

- Do not use `svn checkout`

- Make sure you are not just retrieving the trunk directory

The template as it is will build, so you can try out basic functionality without making any changes.

Step 2: Adjust the package name Replace the 'rtemplate' by the name of your package whereever it is referenced in the code. You may execute a command like

```
find . -type f -exec sed -i 's/rtemplate/<your_package_name>/g' {} +
```

in the command line while in the base directory of your package.

Furthermore, rename the following files, again replacing 'rtemplate' by the name of your package:

- `trunk/rtemplate.Rproj`
- `trunk/R/rtemplate-package.R`

Step 3: Adapt/replace/remove the example code and documentation The package comes with some mandatory code and some example implementations that you may use as a template for your own package. You will have to adapt some of the mandatory code for the needs of your project (see the comments in the code for details). Remove the examples from your package, once you don't need them any longer. Read the comments and instructions in the code before doing so.

Here is an overview on all files that are included in the template's `trunk/` directory:

- `cleanup`
- `cleanup-win`
- `DESCRIPTION` (mandatory - adapt as needed)
- `LICENSE` (mandatory - adapt as needed)
- `NAMESPACE` (mandatory - leave unchanged)
- `.Rbuildignore` (mandatory - add further entries as needed)
- `data-raw/testdata.csv` (delete)
- `inst/extra/run-oq.R` (mandatory - update list of unit tests as needed)
- `inst/doc/RModelCombined20May2016.pptx` (delete)
- `inst/doc/SciSW-R_WK_DOC001SupportDocument.pdf` (delete)
- `inst/doc/rtemplate_introduction.pdf`: a pdf version of the rtemplate vignette (delete - you may want to retain a copy outside your own project, though)
- `man-roxygen/example_repeatdocu.R` (delete)
- `R/examples1.R` (delete)
- `R/examples2.R` (delete)
- `R/exampledata.R` (delete)
- `R/your_package_name-package.R` (mandatory - adapt as needed)
- `rtemplate.Rproj` (for your convenience - no need to adapt)
- `tests/testthat.R` (mandatory - leave unchanged)
- `tests/testthat/test-examples1.R` (delete)
- `tests/testthat/test-examples2.R` (delete)
- `tests/testthat/test-misc.R` (adapt as needed)
- `tools/make-ds.R` (adapt implementation if needed, delete if not used)
- `vignettes/introduction.R` (mandatory - adapt)

There are few more scripts in `utils/` - leave them unchanged (they are for your convenience), but delete the `READMEandDELETEME.txt`.

Do not delete any of the folders that are included in this package template, even if they are currently empty. They will be populated when the package is built.

Step 4: Implement your own project See the following sections for some guidance on how to develop and deploy your new tool. Enjoy!

Package development process

We strongly recommend the use of the `devtools` package which facilitates

- loading the development code from disk, `load_all`
- executing standard R checks, `check`
- building of a source package, `build`

Loading the package is helpful to try out your functions as you develop. The standard R checks (for details, see the Writing R Extensions document [4]) should always be successful prior to committing a version to the subversion repository. This step is also executed as an installation qualification (IQ) step during deployment to MODESIM. Building the package is useful to have a version of the package installed in your home directory for experimental use.

Source control and release process

We strongly recommend to structure the subversion repository as follows:

- `trunk`: contains the last stable release
- `tags/REL-X.Y-Z`: is a copy of the respective stable release which is always created by copying `trunk` to this place whenever a release happens
- `branches/develop`: contains the current development branch which you have checked out in your home directory. This branch gets reintegrated/merged with the `trunk` whenever you do a release.
- `branches/feature-complicated`: Should some feature involve many changes, then this may be split out (as a branch from develop that is later reintegrated with develop).

Hence, we recommend that after you have adapted the rtemplate repository to reflect your package name and serve your high-level needs as described above, you should proceed by creating the `branches/develop` branch from the `trunk` (don't forget to commit your changes to `trunk`!) and proceed implementing your package according to the structure described above.

For details on how to branch and create releases, see the SVN book [5]. In particular, study the chapter on branching and merging. Note that once you reintegrate the `develop` branch into `trunk` for a release, you have to take some extra steps in order to continue development on the `develop` branch. This process is described under "Keeping a Reintegrated Branch Alive".

Tip: To keep your SVN repo clean, have a look at the section on "Ignoring Unversioned Items". Note that ist is good practice **not** to check in any build outputs!

Installation on MODESIM

The installation on MODESIM will be performed by DevIT. The installation process is setup such that it will directly pull the sources from the current stable branch of the subversion, which is the `trunk` branch. To ensure exact traceability the installation procedure will also log the subversion revision number which is a unique integer identifying the exact version.

First, a new ticket for an installation request is needed. For the scheduled MODESIM release the installation on MODESIM is then split in two steps:

1. Installation qualification (IQ)
    - Package download
    - Creation of the FS document, the PDF version of the reference manual => `inst/doc/rtemplate.pdf`
    - Creation of the USR document, the vignette of the package => `inst/doc/introduction.html`
    - Package build
    - Package check wrt to standard R checks (`R CMD check rtemplate-version.tar.gz`)
2. Operation Qualification (OQ)
    - Execution of `inst/extra/run-oq.R` => `rtemplate-OQ.log`

The three documents (rtemplate.pdf, introduction.html and rtemplate-OQ.log) are then stored in PROTON which is a QA system storing all test evidence.

To ensure the above steps run smoothly, you should always ensure that the R package `check` is successful. Moreover, you may emulate the installation procedure above with the two scripts

- `utils/install.sh`
- `utils/oq.sh`

You may need to adapt the variables `RPKG` and `SVN_BASE` at the top of these scripts to your package name and SVN URL location. Then you can execute these steps as

```
module load R
cd utils
./install.sh $R_HOME branches/develop
./oq.sh
```

These steps load the most recent R, then install the develop branch (IQ step) and finally run the OQ step. The test evidence documents will be stored in `/tmp/rtemplate_install`.

Tips & tricks

Inclusion of example data sets Data sets must be provided in .rda data format in the `data` folder. However, if the raw data are in some other format, they can be made available as .rda data sets while the package is being built. See `tools/make-ds.R` for instructions on how to prepare data sets and make them available for internal or external use.

Externally available data sets will be accessible to the user, like the `testdata` data set included in this package:

```
kable(testdata)
```

col1	col2	col3
X	10	-1.5950328
X	9	-0.8101266
y	8	-0.0458064
y	7	-0.8939576
y	6	0.7886874
y	5	0.7215777
z	4	1.9087362
z	3	-0.8860799
z	2	1.5737675
y	1	-0.5493432

Externally available data sets must be documented in an `R` code file. See `R/testdata.R` for an example on how to document data sets.

Data can also be provided for internal use only, e.g. the `retrieve_data` example function in `R/examples2.R` is accessing (and returning) data items from internal data sets that cannot be accessed in the same way as `testdata` above.

Using documentation examples for unit testing It is recommended to execute the examples from the object documentation and test the results as part of the unit tests. This will reduce test writing effort and make sure that the examples are always running properly. See the tests in `tests/testtaht/test-examples1.R` and `tests/testtaht/test-examples2.R` for some examples.

Note: Both, running examples from the documentation and accessing data from datasets that are built by the package may cause issues when building individual parts of the package (e.g. documentation only) or running the tests. To fix these issues, you may need to maually run the `cleanup` script that is provided with the package. Furthermore, you may need to do a `check` of the package rather than just running the tests.

Repeated chunks of documentation If the same bit of documentation needs to be included in the documentation of multiple items, replication can be avoided by using the `@ template` documentation item. See the example in `man-roxygen/example_repeatdocu.R` and `R/examples1.R`.

Inclusion of additional documentation Documentation material that is not built with the package (i.e. material other than object documentation and vignettes) can be provided in `inst/doc/` - like the additional material that is included in this package (this is a hint that you may want to have a look... ;)).

References

[1] Weber S, *SciSW-R (Scientific Software in R)*. 2016; Doc.No: SciSW-R_WK_DOC001, DiARy location: *Cabinets/Dit Asset Repository Systems/S/SciSW-R (Scientific Software in R)/SystemDevelopment & Specifications/Software Development/SciSW-R_WK_DOC001 Support Document* (also included in this package, see `inst/doc/`)
[2] Wickham H, *R packages*. 2015; http://r-pkgs.had.co.nz/
[3] Wickham H, *Advanced R*. 2015; http://adv-r.had.co.nz/
[4] R Development Core Team, *Writing R Extensions*; https://cran.r-project.org/doc/manuals/r-release/R-exts.html
[5] Collins-Sussman B et. al., *Version Control with Subversion*. 2011; http://svnbook.red-bean.com/en/1.6/index.html

R Session Info

```
# Finally, it is good practice to include session info
# information in the vignette:
sessionInfo()
```

```
## R version 3.2.3 (2015-12-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Red Hat Enterprise Linux Server release 6.7 (Santiago)
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.15.1    rtemplate_1.0-0
##
## loaded via a namespace (and not attached):
##  [1] backports_1.0.1 assertthat_0.1  magrittr_1.5    tools_3.2.3
##  [5] htmltools_0.3.5 yaml_2.1.14     Rcpp_0.12.3     stringi_1.1.2
##  [9] rmarkdown_0.9.5 highr_0.6       checkmate_1.7.3 stringr_1.1.0
## [13] digest_0.6.10   evaluate_0.10
```

```
date()
```

```
## [1] "Mon Dec 12 10:11:12 2016"
```