SciSW-R (Scientific Software in R)

Doc.No: SciSW-R_WK_DOC001

Support Document

Author:

Sebastian Weber,
Expert Stats Methodologist
Basel, Switzerland

E-signature and Date:

Weber Sebastian

Digitally signed by Weber Sebastian
DN: SERIALNUMBER=1707068 + CN=Weber Sebastian, OU=PH, OU=people, DC=novartis, DC=com
Reason: I am the author of this document.
Date: 2016.03.02 14:12:22 +01:00

Document change history:

Version	Date *(dd-mmm-yyyy)*	Name of Author	Comments
1.0	01 Mar 2016	Sebastian Weber	Initial document

List of additional reviewers:

Version	Date of Review *(dd-mmm-yyyy)*	Name of Reviewer	Role / Function
1.0	01 Mar 2016	Joanne Valentine	Operational Quality Manager

Table of Contents

1 Purpose and scope of this document

The purpose of this document is to define coding standards to be followed when using the R language for statistical and graphical computing to develop reusable Scientific Software (SciSW) "libraries" in accordance with SOP-7017983: *COAR Analytics and Clinical Submissions (COAR A and CS) Statistical Software: Validation Strategies for Development and Deployment* [P1].

This document explicitly reinforces best coding standards and practices as defined, documented, and verified by the R Core Development Group in the document *Writing R Extensions* [R1] available in CRAN (*Comprehensive R Archive Network*) [R2] and R's build-in run-time verification utility `R CMD check`. The document *Writing R Extensions* [R1] is the primary source for developing all R packages, including those here denoted as *Scientific Software in R*. Secondary sources, e.g., [R4], [R5], [R6], may be used to supplement the material in [R1].

2 R Coding guidelines for Scientific Software

Note: Whenever the wording *should* is used, it is used to indicate that the convention should be used always; however occasional exceptions are allowed, if justified. The words *must* and *never* indicate strict adherence, with no exceptions allowed.

2.1 Development environment

Reusable SciSW code written in R must be organized as an R package. R packages may be developed either locally on a Windows CONNECT laptop/workstation or in the MODESIM environment [R3]. However, the final testing and verification must be done in the MODESIM environment. To create source files users may use their favorite text editor, but they must ensure that the source files they create follow the guidelines described below.

2.2 Text files encodings

For maximal portability, source code (text) files must use ASCII (https://en.wikipedia.org/wiki/ASCII) or UTF-8 (https://en.wikipedia.org/wiki/UTF-8) as character encodings. On Windows, text files are commonly encoded using the Windows 1252 code page (https://en.wikipedia.org/wiki/Windows-1252), and those files may work on UTF-8 environments (like MODESIM), provided only alpha-numeric and strictly ASCII characters are used in the file (for instance, some non-standard characters, like the so-called "smart quotes," cannot be displayed/interpreted under ASCII nor UTF-8 encodings). It is recommended to use automated solutions. For example, subversion allows to set the file property `svn:eol-style` which should be set to the value `native`. This ensures that code checked out on Windows uses Windows style EOL while code checked out on UNIX uses UNIX style EOL.

2.3 Source control system

SciSW must be developed and maintained using a source control system (SCS). Users must choose among the available SCS within the MODESM environment (e.g., subversion or git) or within the GPS environment (Global Programming and Statistical), (e.g., ClearCase).

2.4 Organization of the code

The SciSw must be organized as an R package. See [R1] for details.

R code files must have the extension .R.

2.5 Package vignettes

SciSW packages must provide at least one vignette (this is an extension to [R1], which makes vignettes optional).

A vignette is a high-level document that describes an R package. Unlike `help` documentation that describes low-level computational units (e.g., functions, datasets, objects), vignettes may provide an overview of the package, a tutorial on the use of the package, an extended set of examples, case studies, algorithmic details, etc.

SciSW R packages *must* provide at least one vignette that describes the purpose of the software and a full description of its most important functionality (so-called user requirements) that is being provided. See Section 1.4 in [R1] for details on writing R package vignettes.

SciSW R packages *should* provide additional vignettes to help users with tutorials, detailed examples including output interpretation, etc., to help users understand and start using the SciSW as quickly and completely as possible. Low-level details (including full description of function arguments) must be documented elsewhere as `help` pages (see Section 2 in [R1]).

2.6 Unit, functional, and regression testing scripts

Unit, functional, and regression test scripts must be written and stored in the `tests` folder of the source R package as described in [R1]. All the user-level functions must be documented and tested, but see the alias attribute in Section 2.1.1 *Documenting Functions* in [R1] for documenting and testing closely related functions.

Users may follow equivalent testing strategies provided by contributed CRAN [R2] packages, for instance, `checkpoint`, `RUnit`, and/or `testthat` [R5].

2.7 Package testing and verification

A SciSW R package must pass the utility `R CMD check` without any errors or warnings before it can be approved for installation and use. This utility ensures that R-mandated coding standards are followed, and that all functions, datasets, and other R objects are properly documented (e.g., that all function arguments are described), all examples and testing scripts run without errors and generate the expected outputs. The utility `R CMD check` creates a full and complete description and status of all R-mandated and user-specified tests in a log file named `00check.log`. (See Section 1.3 in [R1].)

2.8 Programming style

A consistent style within a project or software activity is important, since it makes the code more readable and maintainable. In this section, some recommendations are given; for further suggestions, the developer is encouraged to refer to existing, well-established style guides, such as [R1], [R6] or [R7].

The most important rule is: *be consistent*. For example, all functions should be written using the same spacing, indentation rules, and approach to braces, brackets, and parentheses throughout the code.
The following sections present strong suggestions based on commonly used styles.

2.8.1 Line length and semicolons

Lines should be kept within 80 characters and should only occasionally exceed this limit. If line exceeds this limit, the code should be appropriately indented.

Semicolons separating commands should not be used. For clarity, there should be at most one command per line, though a single command might span multiple lines.

2.8.2 Indentation

Indentation must be used to make the code easier to read (indentation highlights the *intended* nesting of code blocks, say, with respect to if/then/else, looping, and other syntactic structures). Two or four spaces should be used for indenting code. Only space characters and *no tabs must be used*.

2.8.3 Spacing

Appropriate spacing should be used to make the code easier to read.

The convention introduced in the Google R Style Guide [**Error! Reference source not found.**7] is recommended, that is:

- Spaces should be placed around all binary operators (=, ==, +, >, <−, etc.). *Exception*: Spaces around the equality signs are optional when passing parameters in a function call.
- A space shouldn't be placed before a comma, but should be placed after a comma.
- A space should be placed before left parenthesis, except in a function call.
- Extra spacing (i.e., more than one space in a row) is allowed if it improves alignment of equals signs or arrows (<−).
- No spaces around code should be placed in parentheses or square brackets. Exception: there should be a space after a comma.

2.8.4 Assignment operator

The assignment operator "<-" should be preferred over the equal symbol "=" to perform assignment.

2.8.5 Function definitions and calls

Function definitions should first list arguments without default values, followed by those with default values.

Functions should be kept short and concise. When a sub-algorithm is used multiple times within different functions, it is recommended to define and use helper functions to support consistent development as well as readability and understanding of the code's logic.

It is recommended to only export a minimal set of functions to the user namespace, i.e. helper functions as described above should only be exported if it is anticipated that the user of the package will benefit from the availability of these functions.

2.8.6 Object names

Descriptive object names should be used to support self-documenting code. For example, variable names `input.file.path`, `model.has.converged` are preferred to abbreviations or acronyms such as `fn`, or `conv`. Function names should start with a verb: e.g., `ComputePower`.

Object names, in general, have lower case letters and words separated with dots (`.`). While dots are preferred, dots, CamelCase, or underscores can separate words in function names. Be consistent: styles should not be mixed.

For (exported) functions residing in other nonstandard, dependent R packages, the double-colon notation should be used whenever possible to clearly indicate which namespace the object belongs to.

2.8.7 References to objects external to the scope of the function

Within a function, references to objects external to the scope of that function call should not be used (e.g., avoid global variables and follow the principle that within functions only local variables and function arguments may be used; functions should return values and not modify the global workspace environment).

`attach()` *must never* be used.

2.8.8 Errors

Errors and warnings should be raised using `stop()` and `warning()`, respectively. Use `try()` and/or `tryCatch()` constructs to manage errors and exceptions.

2.9 Documentation and commenting

2.9.1 Comments

Comments should be used within the function definitions to explain the logical flow of the code. Make sure comments reflect the actual code, in particular make sure comments are updated as the code is modified.

2.9.2 Help files

All user-level functions, object classes, and data objects used must be documented in .Rd files, as required in Section 1.1.3 in [R1]. (This is verified by `R CMD check`.)

3 Procedures, References and Abbreviations/Acronyms/Glossary

3.1 Procedures

	Name	Doc. No. / Vers. No./ Date	Owner / Location
P1	COAR Analytics and Clinical Submissions (COAR A and CS) Statistical Software: Validation Strategies for Development and Deployment	SOP-7017983 Version: Date: 07-Jan-2016	COAR A AND CS / ESOPS

3.2 References

	Name	Doc. No. / Version No. / Effective Date	Owner / Location
R1	Writing R Extensions. R Core Team. 2015.	NA	https://cran.r-project.org/doc/manuals/r-release/R-exts.html
R2	The Comprehensive R Archive Network (CRAN). R Core Team. 2016.	NA	https://cran.r-project.org
R3	MODESIM 101 – Initial User Training. Harvey Bowen. 2015.	Version 4.6	http://modesim101.lampms.eu.novartis.net/modesim101/MODESIM_101_training_document.docx
R4	Software for Data Analysis: Programming with R. John M. Chambers. Springer. 2008.	NA	NA
R5	R Packages. Hadley Wickham. O'Reilly Media Inc. 2015.	NA	NA
R6	Advanced R. Hadley Wickham. CRC Press. 2014.	NA	NA
R7	Google R guide	NA	https://google.github.io/styleguide/Rguide.xml

3.3 Abbreviations / Acronyms / Glossary

Abbreviation / Acronym	Description
CRAN	The Comprehensive R Archive Network
R	R – a statistical and graphical computing language
DiARy	Development Informatics Asset Repository
GPS	Global Programming and Statistics Environment
MODESIM	Modeling and Simulation Environment
SciSW	Scientific Software
SCS	Source Control System

4 Distribution

The original copy of this document is electronic and will be stored in the DiARy system in the folder:

Cabinets/Dit Asset Repository Systems/S/SciSW-R (Scientific Software in R)/System Management