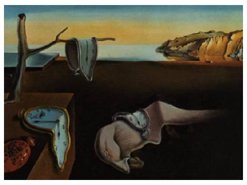


Journal of Statistical Software

June 2012, Volume 49, Issue 9. *http://www.jstatsoft.org/*

RKWard: A Comprehensive Graphical User Interface and Integrated Development Environment for Statistical Analysis with R

Stefan Rödiger
Charité-Universitätsmedizin Berlin

Thomas Friedrichsmeier
Ruhr-University Bochum

Prasenjit Kapat
The Ohio State University

Meik Michalke
Heinrich Heine University Düsseldorf

Abstract

R is a free open-source implementation of the S statistical computing language and programming environment. The current status of R is a command line driven interface with no advanced cross-platform graphical user interface (GUI), but it includes tools for building such. Over the past years, proprietary and non-proprietary GUI solutions have emerged, based on internal or external tool kits, with different scopes and technological concepts. For example, `Rgui.exe` and `Rgui.app` have become the de facto GUI on the Microsoft Windows and Mac OS X platforms, respectively, for most users. In this paper we discuss **RKWard** which aims to be both a comprehensive GUI and an integrated development environment for R. **RKWard** is based on the **KDE** software libraries. Statistical procedures and plots are implemented using an extendable plugin architecture based on ECMAScript (JavaScript), R, and XML. **RKWard** provides an excellent tool to manage different types of data objects; even allowing for seamless editing of certain types. The objective of **RKWard** is to provide a portable and extensible R interface for both basic and advanced statistical and graphical analysis, while not compromising on flexibility and modularity of the R programming environment itself.

Keywords: GUI, integrated development environment, plugin, R.

1. Background and motivation

In mid 1993 Ihaka and Gentleman published initial efforts on the computing language and programming environment R on the *s-news* mailing list. Ambitions for this project were to develop an S-like language without inheriting memory and performance issues. The source code of R was finally released in 1995, and since 1997 development has evolved under the umbrella of the R Development Core Team (R Development Core Team 2001, 2012a; Ihaka 1998). R does not include an advanced cross-platform graphical user interface (GUI) as known from other statistical software packages. However, R includes tools for building GUIs mainly based on Tcl/Tk (Dalgaard 2001, 2002). Meanwhile a plethora of R GUIs have emerged (see Grosjean 2010, for a comprehensive list). In 2005 John Fox released version 1.0 of R Commander (Fox 2005, package **Rcmdr**), which can be considered a milestone in R GUI development; it was the first GUI implementation that was able to make statistical tests, plots and data manipulation easily accessible for R novices. John Fox stated that **Rcmdr**'s target was to provide functionality for basic-statistical courses, though the features have increased over time beyond this (Fox 2005, 2007). In November 2002 Thomas Friedrichsmeier started the **RKWard** open-source software project with the goal to create a GUI for R based on **KDE** (**KDE** e.V. 2012) and **Qt** (Nokia Corporation 2012) technologies [1].

The scope of **RKWard** is deliberately broad, targeting both R novices and experts. For the first group, the aim is to allow any person with knowledge on statistical procedures to start using **RKWard** for their everyday work without having to learn anything about the R programming language, at least initially. At the same time, **RKWard** tries to support users who want to learn and exploit the full flexibility of the R language for automating or customizing an analysis. At the other end of the learning curve, **RKWard** provides advanced integrated development environment (IDE) features to R experts to assist in writing R scripts. Yet, the idea is that R experts too will benefit from the availability of task-oriented GUI dialogs, such as when exploring an unfamiliar type of analysis or by allowing to implement routinely performed tasks as a GUI element. In addition, many features like the integrated data editor and the plot preview will be useful to R novices and R experts alike in their everyday work (see Section 3).

RKWard provides a high level of transparency about the steps that are needed to perform any supported task in R, in order to make it easy for the user to see complete codes for all GUI actions[2]. In doing so, **RKWard** deliberately generates comparatively verbose code. It avoids wrapping complex sequences of data manipulation or analysis into custom high-level R functions. The task of providing high-level functions is logically independent of the development of the GUI frontend, and should best be addressed in dedicated R packages, where necessary. This approach allows to make better use of the modular design of R, avoids locking-in users to a specific GUI application, and provides them with more options for customizing the generated code patterns.

While **RKWard** tries to address users wishing to learn R, it is specifically not designed as a teaching tool – such as **Rcmdr** (Fox 2005) or **TeachingDemos** (Snow 2012) – but as a productive tool. Since its incarnation **RKWard** has gained acceptance for usage in peer-reviewed publications (Zou and Tolstikov 2008, 2009; Rugg-Gunn, Cox, Ralston, and Rossant

[1] **KDE** is a desktop environment and software collection based on **Qt**. In the context of this paper, the term **KDE** is primarily used to refer to the programming library and runtime environment of **KDE**, rather than the complete software collection. For an introduction to **KDE** as a programming library, see Faure (2000). **Qt** is a C++-based cross-platform programming library with a focus on GUI development. For an introduction to programming with **Qt**, see Blanchette and Summerfield (2008).

[2] This distinguishes **RKWard** from R GUIs such as **Red-R** (Parikh and Covington 2010), which specifically aims to hide the complexities of the R programming language, following the concept of visual data-flow programming (Sutherland 1966). In contrast, **RKWard** limits itself to generate R code from GUI settings.

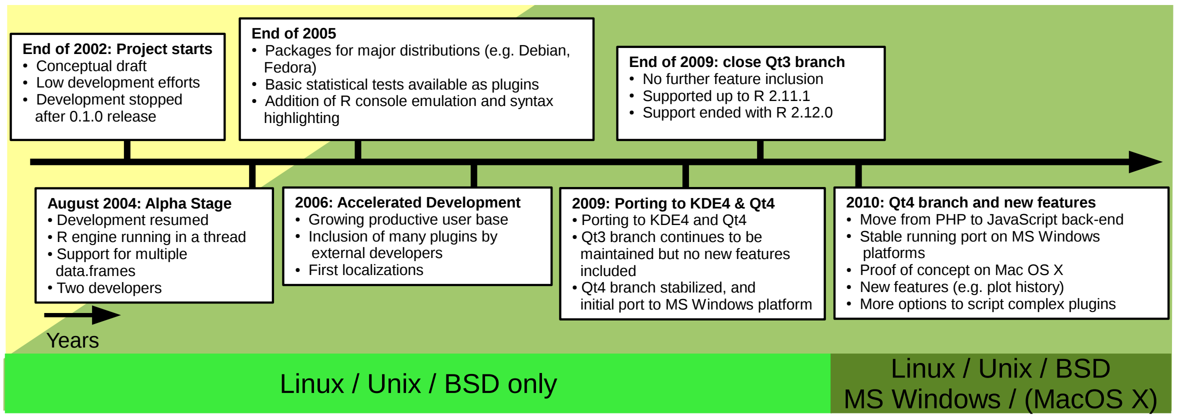

End of 2002: Project starts
• Conceptual draft
• Low development efforts
• Development stopped after 0.1.0 release
End of 2005
• Packages for major distributions (e.g. Debian, Fedora)
• Basic statistical tests available as plugins
• Addition of R console emulation and syntax highlighting
End of 2009: close Qt3 branch
• No further feature inclusion
• Supported up to R 2.11.1
• Support ended with R 2.12.0
August 2004: Alpha Stage
• Development resumed
• R engine running in a thread
• Support for multiple data.frames
• Two developers
2006: Accelerated Development
• Growing productive user base
• Inclusion of many plugins by external developers
• First localizations
2009: Porting to KDE4 & Qt4
• Porting to KDE4 and Qt4
• Qt3 branch continues to be maintained but no new features included
• Qt4 branch stabilized, and initial port to MS Windows platform
2010: Qt4 branch and new features
• Move from PHP to JavaScript back-end
• Stable running port on MS Windows platforms
• Proof of concept on Mac OS X
• New features (e.g. plot history)
• More options to script complex plugins
Years
Linux / Unix / BSD only
Linux / Unix / BSD
MS Windows / (MacOS X)

Figure 1: Timeline of important development milestones and changes in **RKWard**. Time is presented on an arbitrary scale. Here **Qt**3 and **Qt**4 refer to the 3.x and 4.x versions of the **Qt** libraries, respectively and **KDE**4 refers to the 4.x version of the **KDE** libraries.

2010; Yang, Liu, Liu, Qian, Zhang, and Hu 2011; Rödiger *et al.* 2011, 2012). Dialogs for statistical procedures in **RKWard** do not necessarily show a one-to-one correspondence to the underlying steps in R, but are rather oriented at statistical tasks. Furthermore, **RKWard** does not impose artificial limitations on how users can work with the application. For example, the user is not limited to using only one `data.frame` or one model at a time.

RKWard is designed to allow users to create custom GUI dialogs as plugins, requiring relatively little programming knowledge. In essence, **RKWard** plugins consist of an XML file describing the dialog layout, and ECMAScript code which generates R code from the settings made in the GUI. Most of the data handling functionality in **RKWard** is implemented as plugins (see Section 3.5), and many of these plugins have originated as user contributions. Since version 0.5.5, **RKWard** also provides support for downloading user contributed "plugin packs", which are not included in the official **RKWard** releases. Details on the definition of plugins, and a commented example can be found in the technical appendix of this article.

RKWard is licensed under the terms of the GNU General Public License Version 2 or higher. However, due to its dependencies, **RKWard** binaries are effectively distributable only under the terms of Version 2 of the license. Parts of the documentation are available under the GNU Free Documentation License. Full terms and explanations of both licenses are available at `http://www.gnu.org/licenses/gpl.html` and `http://www.gnu.org/licenses/fdl.html`, respectively. While the project remains in constant development, a growing number of users employs **RKWard** in productive scenarios. The source code, selected binaries and documentation is hosted at SourceForge (`http://rkward.sourceforge.net/`). Selected key milestones of the development of **RKWard** are visualized in Figure 1.

In this paper we will give an overview over the installation process (Section 2), the main GUI elements and features of **RKWard** (Section 3), and closing by a short example of a simple **RKWard** session (Section 4). For readers interested in the technical design, and reasons for certain design decisions, a technical appendix of this article is available.

2. Installation and platform availability

Contrary to some other R GUIs, such as **Rcmdr**, **RKWard** cannot be installed and started as a regular R add-on package. Rather, it is started as a stand-alone application which embeds the R engine, and needs to be installed in a platform dependent way, as detailed below[3]. Besides the **KDE** runtime environment and R, **RKWard** utilizes a growing number of R add-on packages. However, these do not have to be installed before hand. Rather **RKWard** will prompt the user to install missing packages interactively, on an as-needed basis (see Section 3.8).

2.1. Installation on the GNU/Linux platform

Historically, **RKWard** originated on the GNU/Linux platform, and binary packages are available for many major GNU/Linux distributions, including Debian, Ubuntu, OpenSuse, Gentoo, Fedora, but also for other POSIX compliant systems such as FreeBSD (`http://standards.ieee.org/develop/wg/POSIX.html`). The exact size of the installation is system dependent. On Debian x86, the package is currently around 1.5 MB (megabyte) compressed, and 5.5 MB uncompressed. However, if the **KDE** runtime environment is not yet installed, an installation of **RKWard** may need several hundred MB of disc space.

On systems which provide up-to-date packages of R and **KDE**, compilation from source is generally unproblematic. Instructions are provided at `http://p.sf.net/rkward/compiling`.

2.2. Installation on Microsoft Windows

RKWard will run on Windows XP, Windows Server 2003, Vista, and Windows 7. 32-bit binaries are provided by the project (download links and instructions are provided at `http://p.sf.net/rkward/windows`). Users can choose between a small installer (1.7 MB), which will add **RKWard** to pre-existing installations of R and **KDE**, and an installation bundle, which provides **RKWard**, R, and **KDE**. This bundle just needs to be unpacked to any user-writable folder, and can be run without any further steps of installation. When using this bundle, **RKWard** can also be installed to removable storage devices (e.g., USB sticks) and shared between systems. Its configuration settings are stored in the user's home directory, and will not be shared across systems, unless the user takes further steps. The size of the current installation bundle is 132 MB compressed, and around 670 MB installed.

Source installation on the Microsoft Windows platform is comparatively difficult, since various tools need to be installed (see `http://sourceforge.net/apps/mediawiki/rkward/?title=RKWard_on_Windows/Packaging` for details).

2.3. Installation on Mac OS X

At the time of writing, the developers lack the resources to support a Mac OS X port, and especially to provide binaries for Mac OS X. Although **RKWard** has been successfully compiled and installed on the Mac, and appeared to be mostly functional, there have also been unresolved reports of failure to compile or start **RKWard** on Mac OS X. Since the **KDE** project currently does not offer binaries for Mac OS X, installation of **RKWard** also requires compilation of the **KDE** runtime environment and its dependencies from source, which takes

[3] See `http://p.sf.net/rkward/download` for an overview and platform specific download links.

many hours to complete on current systems. Further, **RKWard**'s graphics device window related features (see Section 3.6) are only available when compiling and using **KDE** and **RKWard** in **X11** mode. In conclusion, **RKWard** on Mac OS X is not suitable for most users in its current state.

2.4. Starting RKWard

RKWard cannot be loaded from within an R session, but is rather started as a stand-alone application with an embedded R engine. To facilitate the first steps for new users, a dialog offers the choice to load an existing workspace, to start with an empty workspace, or to create a new `data.frame` and open that for editing. Also, an overview help page is shown in the document area of the main window. Both these start-up features can be turned off.

3. Main elements of the user interface

This section gives an overview of the main user interface elements and features of **RKWard**. For a use case oriented example of an **RKWard** session, see Section 4.

The default layout[4] of the main application window is divided into five parts, as depicted in Figure 2. The top of the window is occupied by menu bar and toolbar (Figure 2A). The content of both bars is partially context sensitive, e.g., the `Edit` menu will offer one set of actions when the current document window is a data editor, and another set of actions for a R script editor window. To ease orientation, all top level menus remain persistent, even if no actions are available for that menu in the current context. The menu bar of the main window is also the central access point to most data import, manipulation, analysis, and visualization features (see Section 3.5) for which **RKWard** provides a GUI interface.

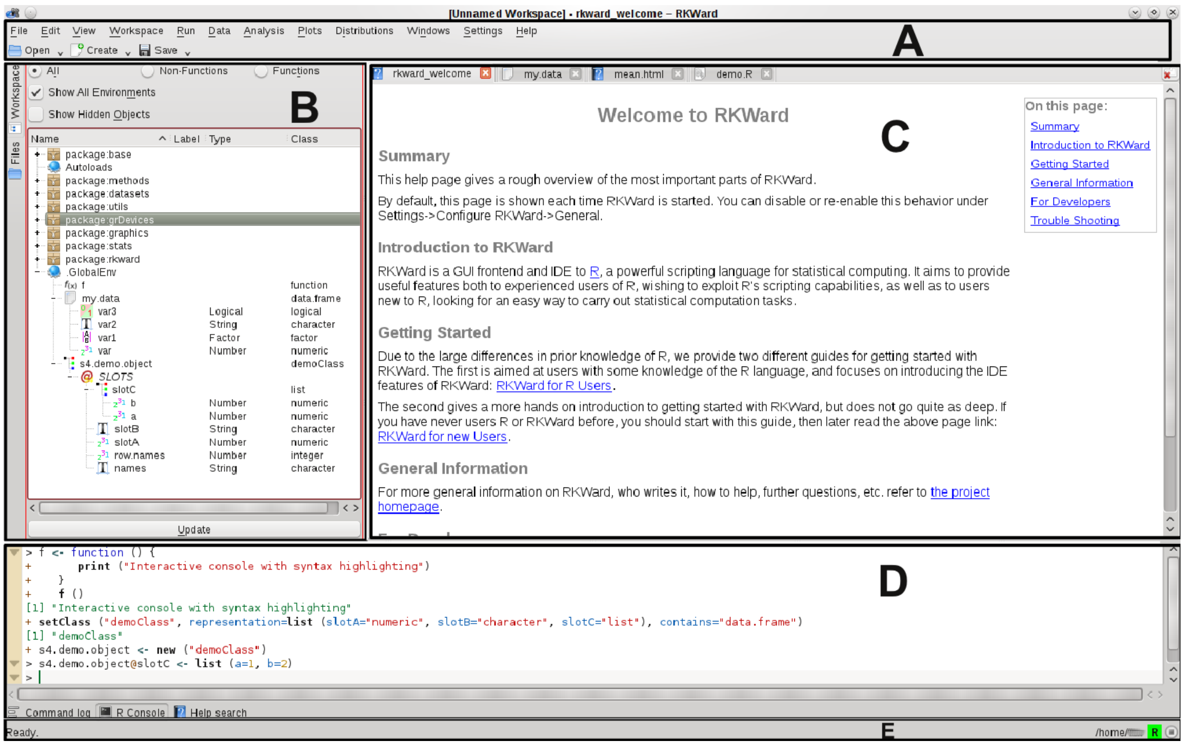
A status bar is shown at the bottom of the window (Figure 2E). It displays (from right to left) a `Stop`-button to interrupt the current computations, the status of the R engine (busy or idle), the current working directory, and a multi purpose region for additional information on some menu items and other GUI elements, visible when hovering the mouse pointer over them.

The **RKWard** GUI generally follows an MDI (multiple document interface) approach. Document windows (object summaries, Section 3.1; script editors, Section 3.2; spreadsheet-like data editors, Section 3.4; results output, Section 3.7; help pages, Section 3.10; and also R on-screen graphics devices, Section 3.6) are arranged in a TDI (tabbed document interface; see e.g., Hopkins 2005; Microsoft Developer Network 2010; Kim and Lutteroth 2009) in the central area (Figure 2C). The order of tabs can be conveniently re-arranged using drag & drop.

Additionally, several tool windows are available form resizable sub-panes at the four sides[5]. By default, the left panel (Figure 2B) contains a file browser (see Section 3.9) and a workspace browser (see Section 3.1), the bottom panel (Figure 2D) contains a command log (Section 3.9), an R console (Section 3.3), and a help search (Section 3.10) window. The top and right sub-panes are not populated by default.

[4] Many aspects of the **RKWard** GUI can be customized by the user. For simplicity we will describe the default appearance of **RKWard**, only.

[5] This combination of a tabbed-document interface and sub-panes is sometimes referred to as an “IDE-style” interface, due to its usage in popular IDEs such as **Eclipse** (Burnette 2005) or **KDevelop** (KDevelop.Org 2011).



```
Name                 Label  Type       Class
+ package:base
  Autoloads
+ package:methods
+ package:datasets
+ package:utils
+ package:grDevices
+ package:graphics
+ package:stats
+ package:rkward
- .GlobalEnv
    f                                  function
  - my.data                            data.frame
      var3                  Logical    logical
      var2                  String     character
      var1                  Factor     factor
      var                   Number     numeric
  - s4.demo.object                     demoClass
    - SLOTS
      - slotC                          list
          b                 Number     numeric
          a                 Number     numeric
        slotB               String     character
        slotA               Number     numeric
        row.names           Number     integer
        names               String     character
```

Welcome to RKWard

Summary

This help page gives a rough overview of the most important parts of RKWard.

By default, this page is shown each time RKWard is started. You can disable or re-enable this behavior under Settings->Configure RKWard->General.

Introduction to RKWard

RKWard is a GUI frontend and IDE to R, a powerful scripting language for statistical computing. It aims to provide useful features both to experienced users of R, wishing to exploit R's scripting capabilities, as well as to users new to R, looking for an easy way to carry out statistical computation tasks.

Getting Started

Due to the large differences in prior knowledge of R, we provide two different guides for getting started with RKWard. The first is aimed at users with some knowledge of the R language, and focuses on introducing the IDE features of RKWard: RKWard for R Users.

The second gives a more hands on introduction to getting started with RKWard, but does not go quite as deep. If you have never users R or RKWard before, you should start with this guide, then later read the above page link: RKWard for new Users.

General Information

For more general information on RKWard, who writes it, how to help, further questions, etc. refer to the project homepage.

```
> f <- function () {
+       print ("Interactive console with syntax highlighting")
+    }
+    f ()
[1] "Interactive console with syntax highlighting"
+ setClass ("demoClass", representation=list (slotA="numeric", slotB="character", slotC="list"), contains="data.frame")
[1] "demoClass"
+ s4.demo.object <- new ("demoClass")
> s4.demo.object@slotC <- list (a=1, b=2)
> 
```

Figure 2: Default **RKWard** main window after start up. (A) Menu bar and toolbar, (B) tool panel showing workspace browser, (C) document view area, showing different documents (welcome message, `data.frame` "my.data", "mean" help page, R script `demo.R`), (D) tool panel showing embedded R console, and (E) status bar with an option to stop running processes. Panels B and D can be resized or collapsed. The red border around B indicates that the workspace browser is the active interface element.

Users can also detach all types of document windows and tool windows from the main application window, which will then appear as independent windows, managed by the window manager, or re-attach them to the main window. This is to allow users to take advantage of an SDI (single-document interface), where useful, such as the ability to view any two documents side-by-side, or to make better use of multiple displays. On-screen graphics device windows are created detached by default, but can be attached to the document view area of the main window.

Windows can be selected (or shown / hidden) using a mouse device with point & click, as well as using a series of keyboard shortcuts (defined by default) for activating specific tool windows, or for cycling through all windows in the order of most recent usage[6].

All key bindings can be configured from the GUI via `Settings`→`Configure Shortcuts`. However, for technical reasons only the shortcuts of currently active components will be listed. Thus, for example, to configure data editor shortcuts, one has to open a data editor first and then to select `Settings`→`Configure Shortcuts`. Since **RKWard** relies on the **RKWard**

[6] This uses the shortcut `Ctrl+Tab` by default, and behaves similar to the `Alt+Tab` feature of common window managers. The difference is that this cycles through **RKWard** windows, only, including both detached windows, and windows which are attached inside the main application window.

editor component, shortcuts for the script editor (Section 3.2) are managed separately via `Settings`→`Configure Editor`→`Shortcuts`. On most systems, it is also possible to configure shortcuts by right-clicking on the respective menu item.

The choice of available actions on the toolbar can be configured via `Settings`→`Configure Toolbars`. Further, it is possible to add and remove sets of data manipulation and analysis features from the GUI, using `Settings`→`Configure RKWard`→`Plugins`.

3.1. Workspace browser and object viewer

The workspace browser (Figure 2B) allows to view and manipulate R objects, similar to a regular file-system browser. This includes both, user objects (data, functions, environments) in `.GlobalEnv` and non-user objects in other environments in the R search path (typically, R package environments). Objects are shown in a hierarchical tree structure. For instance, an object of class `list` can be expanded to show all contained objects by clicking on the + symbol left of the object name. The basic type of each object is indicated by specific icons[7]. Further information on each object can be seen by hovering the mouse pointer over the respective icon. A tooltip window will appear, including information such as dimensionality or function arguments, depending on the type of object. Further, objects inside `.GlobalEnv` can be removed, renamed, and edited from the context menu.

Several actions are available from a context menu (after right-clicking on the object names), depending on the type of object. These allow to search the R help for information on that object, to open a window with detailed information on the object, to delete, rename or copy the object to a new symbol name, or to copy it to `.GlobalEnv`. Further, the context menu allows to open supported types of objects for editing (see Section 3.4; currently, only `data.frame`s can be edited, and only while they exist in `.GlobalEnv`). Selecting `View` from the context menu opens a new window in the document area, containing basic information on the object as well as tabs which show the output of `print()` and `summary()` calls.

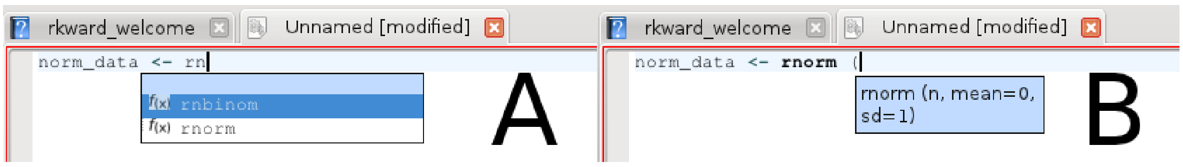
Literally hundreds or even thousands of objects are present in a typical R session. This can be overwhelming at first, therefore, the workspace browser has options to show only a certain subset of objects, e.g., only functions or only data objects, including or excluding hidden objects (object names starting with a "."), or showing only the contents of `.GlobalEnv` as opposed to all environments in the search path.

An object list similar to the workspace browser (but showing only `.GlobalEnv` by default) is also used in several places for the selection of objects to work with, e.g., in an analysis plugin (see Section 3.5).

3.2. Code editor

RKWard comes with an advanced R script editor, based on the **KDE** advanced text editor component (**Kate**; `http://kate-editor.org/`). Features of this editor include syntax highlighting (both on screen and in printouts; for R and many other script types), code folding, block-wise indentation adjustments or commenting, automatic brackets, search and replace with plain text or regular expressions, and many more. Further, **Kate** can be extended by customized actions implemented in ECMAScript (Haumann 2010). The editor automatically

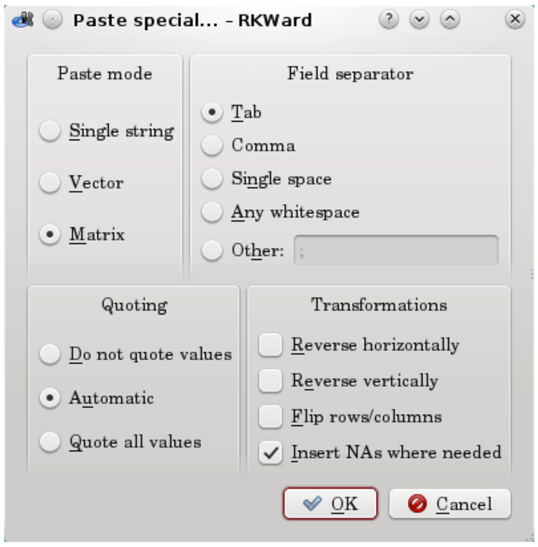
[7]The workspace browser indicates the types "Number", "Factor", "String", and "Logical" for the `data.frame` "my.data" (Figure 2B).



```
norm_data <- rn
rnbinom
rnorm
```

```
norm_data <- rnorm (
rnorm (n, mean=0, sd=1)
```

Figure 3: Code hinting features of the script editor. The script editor is able to hint (A) R object names and (B) function arguments.



Paste mode: Single string, Vector, [✓ Matrix]

Field separator: [✓ Tab], Comma, Single space, Any whitespace, Other: ;

Quoting: Do not quote values, [✓ Automatic], Quote all values

Transformations: Reverse horizontally, Reverse vertically, Flip rows/columns, [✓ Insert NAs where needed]

Figure 4: Paste special dialog. This tool allows to paste data (e.g., tabular, text) from the clipboard, directly to an R script and therefore accelerates the work process with data from different sources like spreadsheet applications.

saves snapshots of the currently edited files at configurable intervals.

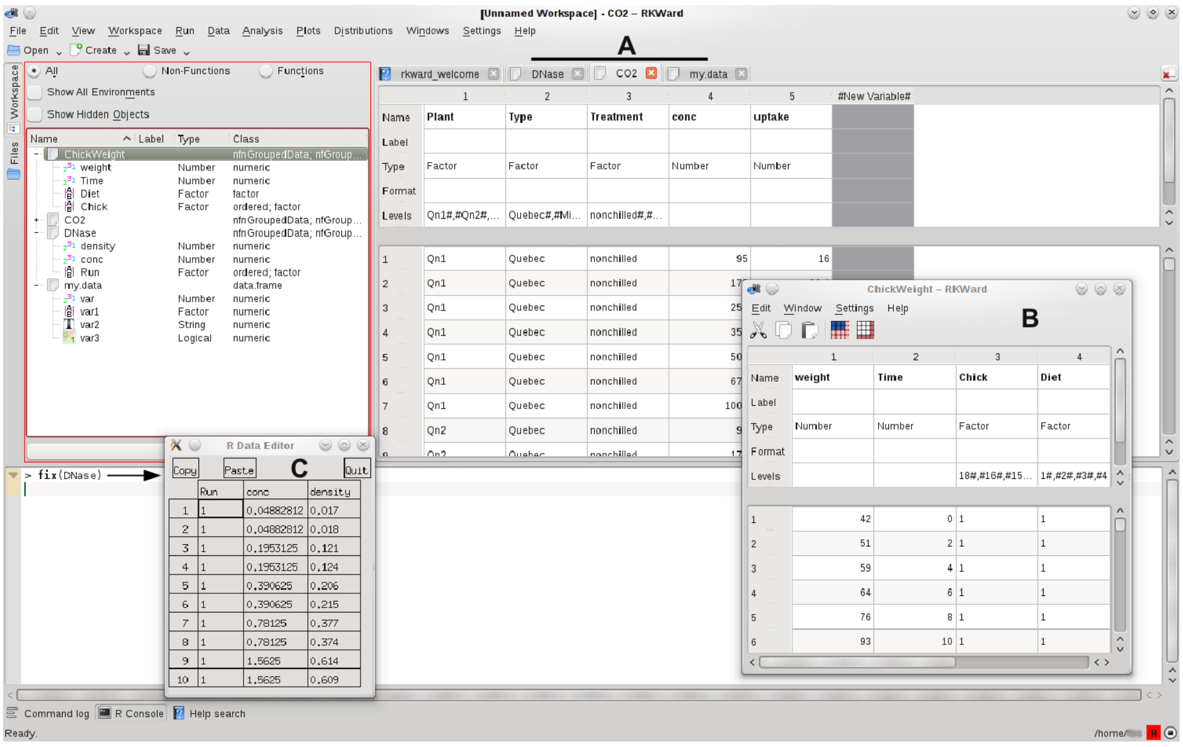
For interaction with R, the editor has predefined shortcuts (and toolbar icons) for submitting the current line, the current selection, predefined blocks, or the entire document to the R engine for evaluation. It also offers object name completion and function argument hinting (Figure 3A and B) based on the objects present in the R workspace[8]. A further feature specific to the R language is the `Paste Special` action, which allows to paste the clipboard content (e.g., from a separate spreadsheet application) as a single string, vector, or matrix, suitable for inclusion in an R script, optionally transforming it in advance (Figure 4).

Script editor windows can be created by opening an existing R script file from the file browser or the `File` menu. It can also be invoked from R, e.g., using the `file.edit()`, `file.show()`, or `fix()` commands.

3.3. Using the R console

For users with knowledge of R, **RKWard** provides direct access to the embedded R engine in the R console tool window. It is important to understand that technically this is an emulation

[8]The object name completion and function argument hinting features in **RKWard** predate the inclusion of similar features into the core R distribution. For this reason, they are technically based on different mechanisms.


[Unnamed Workspace] - CO2 – RKWard
File Edit View Workspace Run Data Analysis Plots Distributions Windows Settings Help
Open Create Save
A
rkward_welcome DNase CO2 my.data
Workspace
Files
All Non-Functions Functions
Show All Environments
Show Hidden Objects
ChickWeight – RKWard
Edit Window Settings Help
B
R Data Editor
Copy Paste Quit
C
Command log R Console Help search
Ready.
/home/

Name	Label	Type	Class
ChickWeight			nfnGroupedData; nfGroup…
weight		Number	numeric
Time		Number	numeric
Diet		Factor	factor
Chick		Factor	ordered; factor
CO2			nfnGroupedData; nfGroup…
DNase			nfnGroupedData; nfGroup…
density		Number	numeric
conc		Number	numeric
Run		Factor	ordered; factor
my.data			data.frame
var		Number	numeric
var1		Factor	numeric
var2		String	numeric
var3		Logical	numeric

	1	2	3	4	5	#New Variable#
Name	Plant	Type	Treatment	conc	uptake	
Label						
Type	Factor	Factor	Factor	Number	Number	
Format						
Levels	Qn1#,#Qn2#,…	Quebec#,#Mi…	nonchilled#,#…			
1	Qn1	Quebec	nonchilled	95	16	
2	Qn1	Quebec	nonchilled	17		
3	Qn1	Quebec	nonchilled	25		
4	Qn1	Quebec	nonchilled	35		
5	Qn1	Quebec	nonchilled	50		
6	Qn1	Quebec	nonchilled	67		
7	Qn1	Quebec	nonchilled	100		
8	Qn2	Quebec	nonchilled	9		
9	Qn2	Quebec	nonchilled	17		

	1	2	3	4
Name	weight	Time	Chick	Diet
Label				
Type	Number	Number	Factor	Factor
Format				
Levels			18#,#16#,#15…	1#,#2#,#3#,#4
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
4	64	6	1	1
5	76	8	1	1
6	93	10	1	1

```
> fix(DNase)
```

	Run	conc	density
1	1	0.04882812	0.017
2	1	0.04882812	0.018
3	1	0.1953125	0.121
4	1	0.1953125	0.124
5	1	0.390625	0.206
6	1	0.390625	0.215
7	1	0.78125	0.377
8	1	0.78125	0.374
9	1	1.5625	0.614
10	1	1.5625	0.609

Figure 5: **RKWard** with several `data.frame`s in use at the same time. (A) One `data.frame` (`CO2` data of the **datasets** package) is opened for editing in the main window. Two further `data.frame`s are opened in the background in tabs. (B) Another `data.frame` (`ChickWeight`) is opened in a detached window. (C) R's standard data editing features (e.g., `fix()`, `edit()`) are also usable in **RKWard**. In this example `fix(DNase)` was invoked from the console (arrow).

of R running in a console session, not a real R session. This leads to a few subtle differences, e.g., with respect to the command history feature in R.

However, for most purposes **RKWard**'s R console can be used exactly like R running in a terminal. Adding to that, it provides many of the features which are also available in the code editor (see Section 3.2). Most prominently, it supports syntax highlighting, code folding, function argument hinting, object name completion, and pasting vector or matrix data directly from the clipboard. By default, any code that is submitted to the R engine from the code editor or from help pages, is sent through the R console. However, it can be configured to be submitted in the background, instead.

3.4. Spreadsheet-like data editor

Historically, one of the earliest features of **RKWard** is a built-in spreadsheet-like data editor. Currently, editing R objects of type `data.frame` is possible. In contrast to the `data.frame` editing shipped with the R core distribution, this editor gives the illusion of "in-place" editing of data. New `data.frame`s can be created and opened from the GUI, and existing objects can be opened for editing from the workspace browser. For opening objects from R code, the function `rk.edit()` can be used. Figure 5 shows multiple `data.frame`s open for editing.

Metadata on each column of a `data.frame` (i.e., name of the column, data type, and potentially data labels) is shown in the upper portion of the data editor, and can be manipulated there, while the data itself is shown in the lower portion. The upper portion can be hidden using a slider, to save space for the display and editing of actual data. Similarly, an editable column showing the row names of the `data.frame` can be shown or hidden separately from the data.

For columns of type `factor`, factor levels can be edited by double-clicking on the `Levels` row of the meta information. Levels can also be assigned to other types of variables, but only for consecutive integer values. These levels will be displayed instead of the underlying value, if applicable. Each column can also be assigned an arbitrary descriptive "Label", which is stored in R as an attribute of the column.

Contrary to many other editors, the data editor in **RKWard** does not automatically convert data types of columns. For instance, if a non-numeric string is entered into a cell of a numeric column, the data type of the column remains numeric, and the entered value is highlighted in red. Internally, the invalid cell is set to `NA`. The entered value is stored separately, in an attribute of the column. The rationale for this approach is that it offers protection against accidental, and probably undetected, conversion of data types. The user can manually convert the storage mode of a column by simply selecting a different data type in the "Type" row of the meta information.

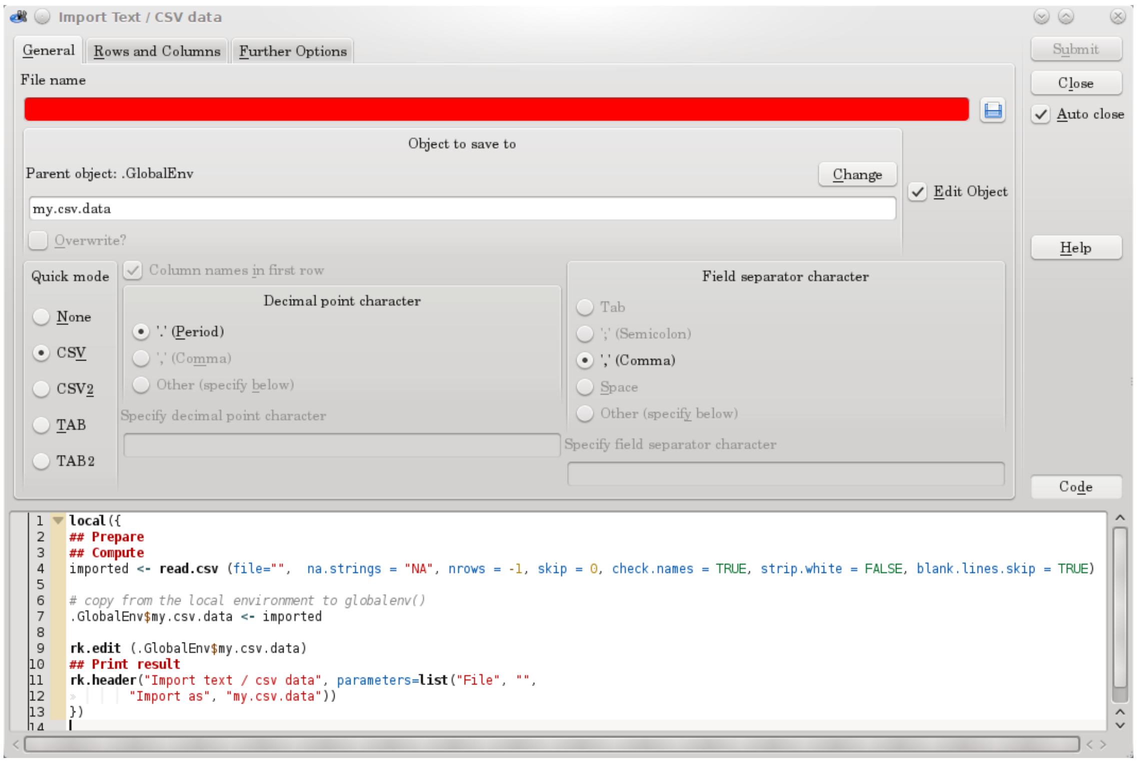
The data editor supports insertion and deletion of rows or columns at arbitrary positions. Rows (columns) can also be added at the bottom (right) by simply entering data into the trailing row (column) shown in gray. Copy & paste is supported, where the area affected by paste operations can optionally be constrained to the selected region, or to the dimensions of the table. The data editor can also be set to read-only mode to examine data objects.

In the context of data editing, it is noteworthy that **RKWard** supports working with multiple objects simultaneously, rather than limiting actions to a single active `data.frame`, as with e.g., **Rcmdr** or **Deducer** (Fellows 2012). Given this non-modal interface design, multiple data editor windows can be opened at once (Figure 5).

3.5. Handling, manipulating, and analyzing data

Dealing with data – i.e., importing, transforming, filtering, analyzing, and visualizing – is the core strength of R, and one central goal of **RKWard** is to make the most of this functionality available to a broader audience by providing it in the form of easy to use GUI dialogs. Since the data handling functionality itself is provided by R and its numerous add-on packages, this can basically be accomplished by defining GUI dialogs, generating R code according to the settings made in the GUI, and having the generated code evaluated by the R engine. This general pattern, implemented as plugins, is the basic recipe for most of the functionality provided by **RKWard** (see the technical appendix of this article for details on the definition of plugins). For the purpose of this article we will look at the standard elements of data handling functions by an example of importing comma-separated values (CSV) data[9]. Further examples are given in Section 4.

[9] Note that on purpose, **RKWard** does not have its own file format for data import and export, but rather uses R workspaces as default data format. Additionally, it is possible to import data from several sources as described in this section. Of course, further formats can also be imported using copy & paste (see Sections 3.2 and 3.4), or by manually entering appropriate R commands in the R console (Section 3.3).



File name:

Object to save to

Parent object: .GlobalEnv

my.csv.data

[✓ Edit Object]

Overwrite?

Quick mode: None [✓ CSV] CSV2 TAB TAB2

[✓ Column names in first row]

Decimal point character: [✓ '.' (Period)] ',' (Comma) Other (specify below)

Specify decimal point character

Field separator character: Tab ';' (Semicolon) [✓ ',' (Comma)] Space Other (specify below)

Specify field separator character

```
local({
## Prepare
## Compute
imported <- read.csv (file="",  na.strings = "NA", nrows = -1, skip = 0, check.names = TRUE, strip.white = FALSE, blank.lines.skip = TRUE)

# copy from the local environment to globalenv()
.GlobalEnv$my.csv.data <- imported

rk.edit (.GlobalEnv$my.csv.data)
## Print result
rk.header("Import text / csv data", parameters=list("File", "",
          "Import as", "my.csv.data"))
})
```

Figure 6: General data import dialog. Useful defaults for a variety of formats can be set using the `Quick Mode` selector on the left. Further customizations can be done from the `Rows and Columns` and `Further Options` tabs. The code in the bottom area can be copied and used for other purposes.

At the time of writing, **RKWard** provides support for importing SPSS, Stata, and "delimited text" data. Internally, **RKWard** relies on standard R functions and the package **foreign** (Murdoch 2002) for reading these data files. To import CSV data, select `File`→`Import format`→`Import Text`→`CSV` data from the menu. This will open the dialog shown in Figure 6. The central area of this dialog provides options to control the import. The `File name` field is highlighted, to indicate that it is required to specify a file before the dialog can proceed. Further options are available from the tabbed pages of the central area.

The right-side area is common to all data handling dialogs. Here the `Submit` button is used to start the import action. It is enabled once all required settings have been made, i.e., in this case, once a file name has been selected. The `Close` button will close the dialog without taking any action.

The bottom area optionally shows the R code corresponding to the current settings which will be run upon pressing the `Submit` button (see Section 4.1 for generated R code). The code display is hidden by default and can be revealed using the `Code` button. This generated code display is updated dynamically as the user changes settings, allowing to see the effect of each change instantly.

Most data handling functions will produce some output, which is sent to the output window. From there it is possible to repeat the action by clicking on the `Run again`-link (see Section 3.7).

3.6. Graphics window and plot previews

For plotting, **RKWard** relies on the graphics capabilities provided by R. All R devices, including on-screen devices, can be used in the regular way. However, for the `X11()` and `windows()` devices, **RKWard** adds a menu bar and a toolbar to the device windows (on the Microsoft Windows platform, replacing the default menu bar provided by the device). The menu bar and toolbar give access to a number of different functions, including GUI dialogs for exporting the current plot, and adding a grid to an existing plot (works on only certain types of plots).

Further, a history mechanism is provided, which stores created plots automatically and allows to navigate back to earlier plots (Figure 7). The history is available as a drop-down list of the plot calls as well as using typical `back` and `forward` buttons on the toolbar. The maximum number of plots to record, as well as the maximum size of each individual plot, is configurable from the settings menu. This plot history is shared between all open on-screen device windows, yet they behave independently. For example, if multiple devices display the same plot, any modification (including deletion) of the plot on one device renders its instances on other devices as "new" and hence can be added back to the plot history. In addition, duplicating or closing a device window records any unsaved plots to the history.

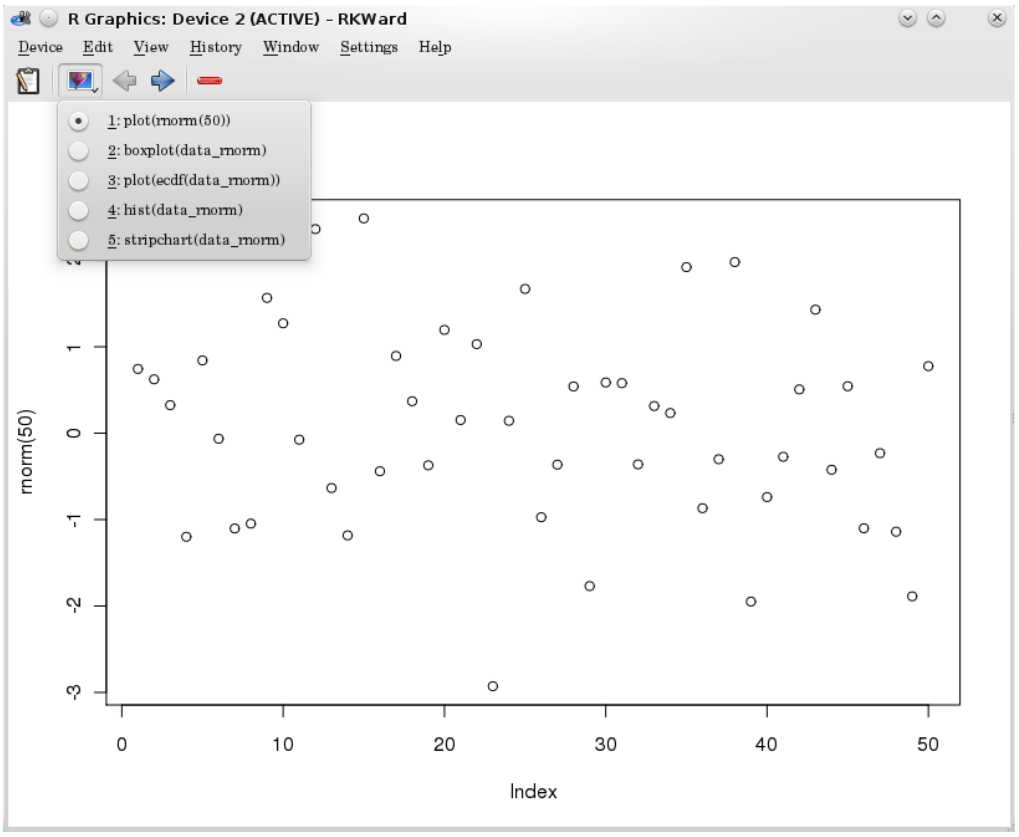


Figure 7: On-screen graphics device window in **RKWard**. The plot history is available as a drop-down list, allowing to jump directly to a previous plot. In this example, five different plots were performed on the same data set of a random sample (`rnorm()`). The plot can be exported via `Device`→`Export` as described in Section 4.3.

Further, **RKWard** provides access to different plotting functions using GUI dialogs, available from the `Plots` menu. Wherever appropriate, **RKWard** supports a "plot preview" feature. When the `Preview` box of the respective dialog is checked, a device window is opened, which shows the plot as it would be created with the current settings (see Section 4.3 for an example). The preview is updated automatically as the user makes changes, allowing to see the effect of each setting instantly[10]. For example, the central limit theorem plugins under the `Distributions` menu can be very helpful to dynamically "show" the convergence in distribution while teaching. For the sake of simplicity, such preview plots are not added to the history.

3.7. Results output

While all basic mechanisms of capturing and documenting R output can also be used, **RKWard** provides a dedicated output file and a output window for documenting the results. All GUI-driven data handling functions (see Section 3.5) write their output to this file. The same applies to error messages, in case a plugin fails to perform its task. The output is presented in a journal format[11]. All results are presented sequentially with the last performed task at the bottom. It is also possible to write to the output directly from R scripts by using a number of dedicated R functions included in the `rkward` package (part of **RKWard**). For the GUI-driven data handling functions, the output is standardized to include the name of the feature, the date and time of its execution, and other basic parameters, wherever applicable. Further, a clickable `Run again` link is rendered below the output of each data handling feature, which allows to invoke the same feature again with identical parameters[12] (see Figure 8). Thus, the `Run again` feature combines the documentation of the result with an automated way to conduct the same analysis again on new data, providing benefits similar to, for example, the automated report generation available from **RReportGenerator** (Raffelsberger *et al.* 2008).

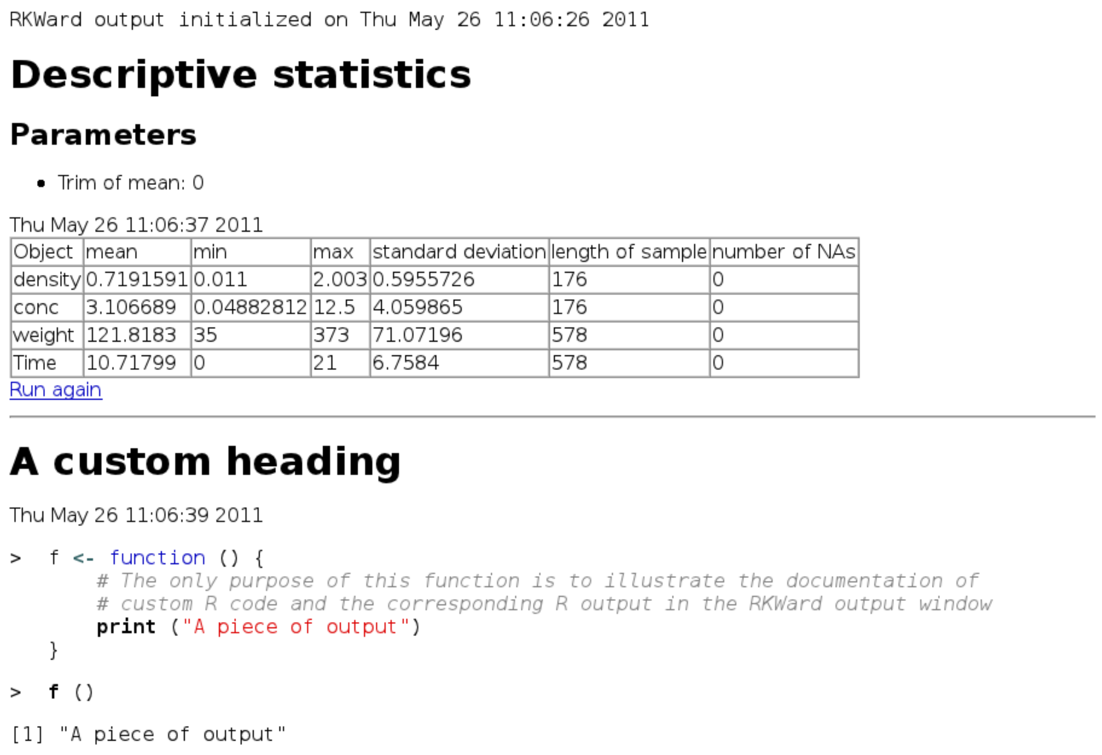
The formatting of output is kept to a minimum. In particular, **RKWard** is very reluctant to round numerical results for the sake of a pretty output. Rather, the focus is on making the results easily accessible for further processing, typically in a dedicated word processor. Output is based on HTML, and the raw HTML file and any images therein can be directly retrieved from a dedicated folder (by default, this is a folder named `.rkward` inside the user's home folder). It is also possible to select and copy sections of the output directly from the output window, and to paste them into office applications as richly formatted text; even images and tables can be easily copied by drag & drop to many office applications. In future releases, it is planned to integrate **RKWard** with existing office suites. This will possibly also mean addition of different file formats such as Open Document Format and technologies such as `Sweave` and **odfWeave** (Leisch 2002; Kuhn 2006).

Images contained in the output are stored as portable network graphics (PNG; `http://www.libpng.org/pub/png/`) by default, but JPEG (`http://www.jpeg.org/jpeg/index.html`) and scalable vector graphics (SVG; `http://www.w3.org/Graphics/SVG/`) can also be used. Similarly, the size of images can be configured by the user. It is expected that SVG will become the default output format eventually, but currently some SVG files produced by R

[10]The preview is updated asynchronously to keep the GUI responsive; see also the technical appendix of this article.

[11]Note: The font size of the output can be adjusted from the menu.

[12]In case not all parameters could be reused, e.g., because some of the objects in question are no longer available, the user will be notified.


RKWard output initialized on Thu May 26 11:06:26 2011

Descriptive statistics

Parameters

- Trim of mean: 0

Thu May 26 11:06:37 2011

Object	mean	min	max	standard deviation	length of sample	number of NAs
density	0.7191591	0.011	2.003	0.5955726	176	0
conc	3.106689	0.04882812	12.5	4.059865	176	0
weight	121.8183	35	373	71.07196	578	0
Time	10.71799	0	21	6.7584	578	0

Run again

A custom heading

Thu May 26 11:06:39 2011

```
>   f <- function () {
        # The only purpose of this function is to illustrate the documentation of
        # custom R code and the corresponding R output in the RKWard output window
        print ("A piece of output")
    }

>   f ()

[1] "A piece of output"
```

Figure 8: Sample contents of the output window. Upper portion: Result of analyzing sample data (from the `DNase` and `ChickWeight` datasets of the **datasets** package) in the "Descriptive Statistics" plugin. Standard elements of plugin output include a standardized header, and a `Run again`-link, which allows to repeat the analysis with identical or similar parameters. Lower portion: A custom heading added using the `rk.header()` function, and a short transcript of R code with corresponding output.

are not properly rendered by older supported versions of the **KDE** libraries.

Users can also add custom content to the output window using `rk.header()`, `rk.print()`, and some related functions. Further, custom R code as well as the corresponding R output can easily be documented in the **RKWard** output window, including syntax highlighting (see the lower portion of Figure 8).

3.8. Package management

The number of R packages available from the Comprehensive R Archive Network (CRAN), **Omegahat** (`http://www.omegahat.org/`) and **Bioconductor** (Gentleman *et al.* 2004) has grown exponentially since R 1.3.0 (2001) to R 2.7.0 (2008) (Fox 2008; Ligges 2003; Visne *et al.* 2009). **RKWard** utilizes functionality from a growing number of these packages, but avoids making the installation of all supported packages a pre-requirement to using **RKWard** at all. Only once a not yet installed package is required to conduct a certain action, a package management dialog is invoked automatically, which allows to download and install the package from a repository such as CRAN. The package management dialog can also be invoked manually from the menu (`Settings`→`Configure Packages`) for installing new or updating existing R packages. The underlying package management technology is that of R (Ligges 2003; Ripley 2005).

RKWard supports installing packages to any user writable location. If no current library location is user writable, **RKWard** offers to create a new one. On Unix systems, interactively acquiring root privileges for installation to the system-wide libraries is also supported. The installation process itself can be monitored at the interface for error tracking. At the time of writing, **RKWard** has no built-in tools for the interactive exploration of R packages. However, it is possible to invoke external helpers as reported elsewhere (Zhang and Gentleman 2004).

3.9. Further tool windows

The file browser tool window can be used to open supported file types (e.g., R scripts, HTML files) inside the main **RKWard** window. For unsupported file types (such as portable document format; PDF), the system's default external applications are used.

The command log window contains a log of the commands that have been evaluated by the R engine, and any output produced by these commands. By default, the log shows only commands which have been entered by the user or directly correspond to user actions, but it can be configured to include commands which are run for **RKWard**'s internal purposes such as keeping the workspace browser up to date.

Commands can be submitted while the R engine has not yet started, or while another lengthy calculation is still in progress. In these cases, commands are placed into a queue first, and executed as soon as the R engine becomes available. The `pending jobs` window (not shown in the tool area by default) lists current R commands waiting for evaluation by the R engine. While this window is mostly of interest to application developers for diagnostic purposes, it can also be used to interrupt selected commands.

3.10. Help system

RKWard provides access to both R specific and **RKWard** specific help pages. R specific documentation includes help pages for functions and packages and the various R manuals. **RKWard** specific documentation consists of help pages on **RKWard** in general and on specific GUI dialogs[13]. All these various types of help pages can be browsed in the same document window, and can be cross-linked. For example, help pages for **RKWard** GUI dialogs will typically link to documentation for both related **RKWard** dialogs and the underlying R functions. An arbitrary number of help windows can be browsed simultaneously, in the TDI view area (see Figure 2C) or in detached windows.

A central access point to the help system is the `Help` menu. Further, help pages on **RKWard** GUI dialogs can be accessed from the dialog itself using the `Help` button. A useful ("reverse") feature here is that these pages include a link near the top of the page to start the corresponding GUI dialog directly. Help on R specific functions can be invoked from multiple places, such as, the context menu of the workspace browser, by pressing F2 (function reference) while the cursor is on a function name either in the code editor or in the R console, and of course, by using the R `help()` command. In addition, a tool view window is provided as an interface to the `help.search()` command in R. This allows to search all installed, all loaded, or specific R packages for a specified topic.

[13]For technical background of **RKWard** GUI help pages please refer to the technical appendix of this article.

The help browser window is based on the **KDE** HTML viewer component and supports many standard features like increasing or decreasing the font size and searching text within a page. Additionally, R code inside a help page can be sent to the R engine for evaluation by selecting it and pressing F8 (or via `Run`→`Run Selection`).

4. Using RKWard: An example session

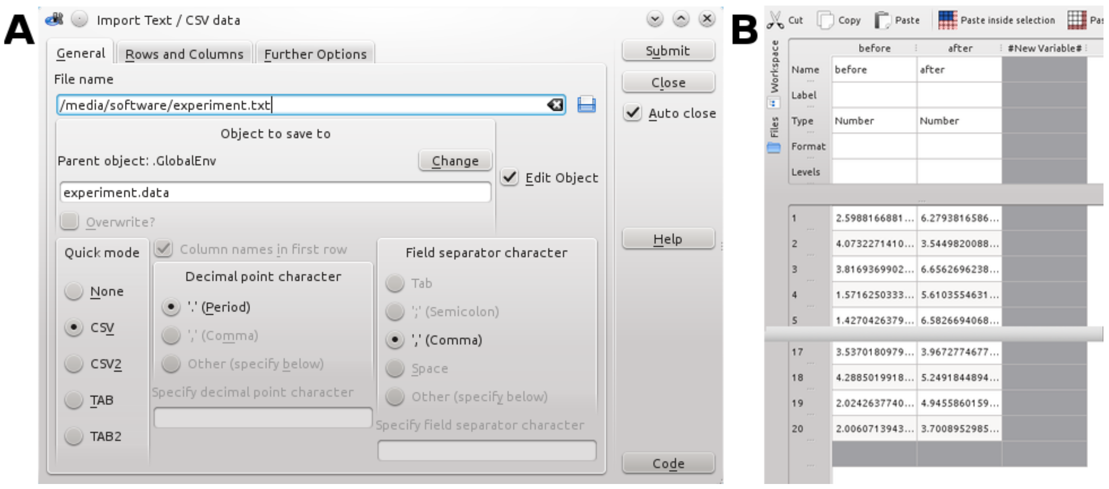
This section describes an example **RKWard** session, in order to give an idea of what working with **RKWard** is like in practice. The session is organized along the routine tasks of importing, analyzing, and visualizing data. In this example, it is assumed that an experimental treatment was given to 20 test subjects. The objective is to compare the responses before and after the treatment.

4.1. Importing data

Suppose that the data was saved as or exported to CSV format, for example, from a spreadsheet application. **RKWard**'s import plugin can comfortably read it into a new R object. The import dialog (`File`→`Import`→`Import format`→`Import Text/CSV data`; Figure 9A) assists in reading the data by a common point & click interface. In this example, "comma" and "period" were chosen via `Quick mode` as the field separator and decimal point characters respectively. The generated R code can be revealed by clicking on the `Code` button:

```
read.csv(file = '/media/software/experiment.txt',
         na.strings = 'NA', nrows = -1, skip = 0,
         check.names = TRUE, strip.white = FALSE, blank.lines.skip = TRUE)
```

Checking the `Edit Object` box automatically opens a data editor tab showing the imported data (Figure 9B).



File name: /media/software/experiment.txt

Object to save to
Parent object: .GlobalEnv [Change]
experiment.data
[✓ Edit Object]
Overwrite? [ ]
[✓ Auto close]

Quick mode: None [✓ CSV] CSV2 TAB TAB2
[✓ Column names in first row]
Decimal point character: [✓ '.' (Period)] ',' (Comma) Other (specify below)
Specify decimal point character:
Field separator character: Tab ';' (Semicolon) [✓ ',' (Comma)] Space Other (specify below)
Specify field separator character:

	before	after	#New Variable#
Name	before	after	
Label			
Type	Number	Number	
Format			
Levels			
1	2.5988166881...	6.2793816586...	
2	4.0732271410...	3.5449820088...	
3	3.8169369902...	6.6562696238...	
4	1.5716250333...	5.6103554631...	
5	1.4270426379...	6.5826694068...	
17	3.5370180979...	3.9672774677...	
18	4.2885019918...	5.2491844894...	
19	2.0242637740...	4.9455860159...	
20	2.0060713943...	3.7008952985...	

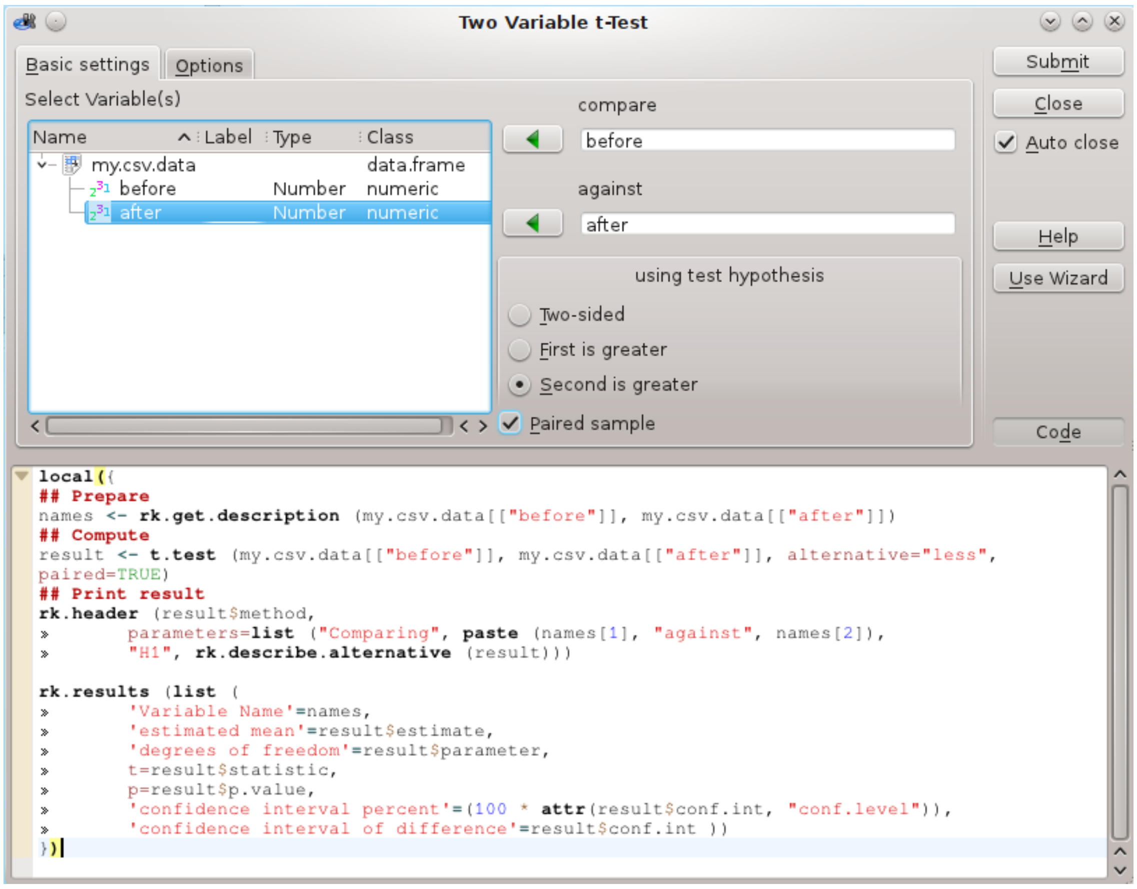
Figure 9: (A) CSV import dialog. Useful defaults for a variety of common text separated value formats can be set using the `Quick Mode` selector on the left. Beyond that, many options can be customized. (B) Data editor. The imported CSV data from `experiment.txt` are presented (data visually trimmed).

4.2. Conducting a Student's t test

To test the hypothesis that the given treatment significantly increased the response, a Student's t test for a paired sample is conducted using the `Analysis`→`Means`→`t-Tests`→`Two variable t-Test` plugin. In the object browser on the left side, the two variables from the expanded R object containing the table of imported data are selected (Figure 10). Pressing the `Submit` button conducts the test, and opens the output document tab showing the results, including the date of analysis and relevant test parameters (Figure 11).

4.3. Creating a plot

To visualize the data, `Boxplot` is chosen from the `Plots` menu and the two variables, corresponding to the Student's t test above, are selected. The dialog allows to define custom variable labels (Figure 12). Checking the `Preview` box opens a graphics window showing the boxplot as it is configured, and updates the window in real time on any changes to plot parameters. The plot can also be exported to several image formats, directly from the preview window (Figure 13).



Select Variable(s)

Name	Label	Type	Class
my.csv.data			data.frame
before		Number	numeric
after		Number	numeric

compare: before

against: after

using test hypothesis

Two-sided

First is greater

[✓ Second is greater]

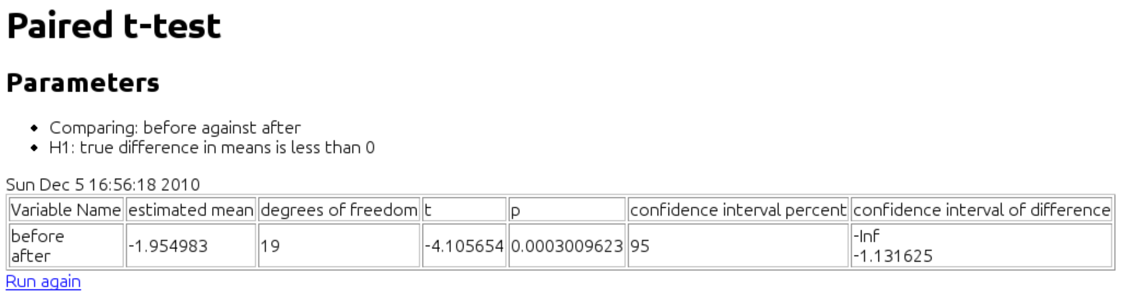
[✓] Paired sample

[✓] Auto close

```
local({
## Prepare
names <- rk.get.description (my.csv.data[["before"]], my.csv.data[["after"]])
## Compute
result <- t.test (my.csv.data[["before"]], my.csv.data[["after"]], alternative="less",
paired=TRUE)
## Print result
rk.header (result$method,
»       parameters=list ("Comparing", paste (names[1], "against", names[2]),
»       "H1", rk.describe.alternative (result)))

rk.results (list (
»       'Variable Name'=names,
»       'estimated mean'=result$estimate,
»       'degrees of freedom'=result$parameter,
»       t=result$statistic,
»       p=result$p.value,
»       'confidence interval percent'=(100 * attr(result$conf.int, "conf.level")),
»       'confidence interval of difference'=result$conf.int ))
})
```

Figure 10: Student's t test dialog for two variables. The bottom area shows the R code corresponding to the settings.



Paired t-test

Parameters

- Comparing: before against after
- H1: true difference in means is less than 0

Sun Dec 5 16:56:18 2010

Variable Name	estimated mean	degrees of freedom	t	p	confidence interval percent	confidence interval of difference
before after	-1.954983	19	-4.105654	0.0003009623	95	-Inf -1.131625

Run again

Figure 11: Results of conducting a Student's *t* test in tabular HTML format.

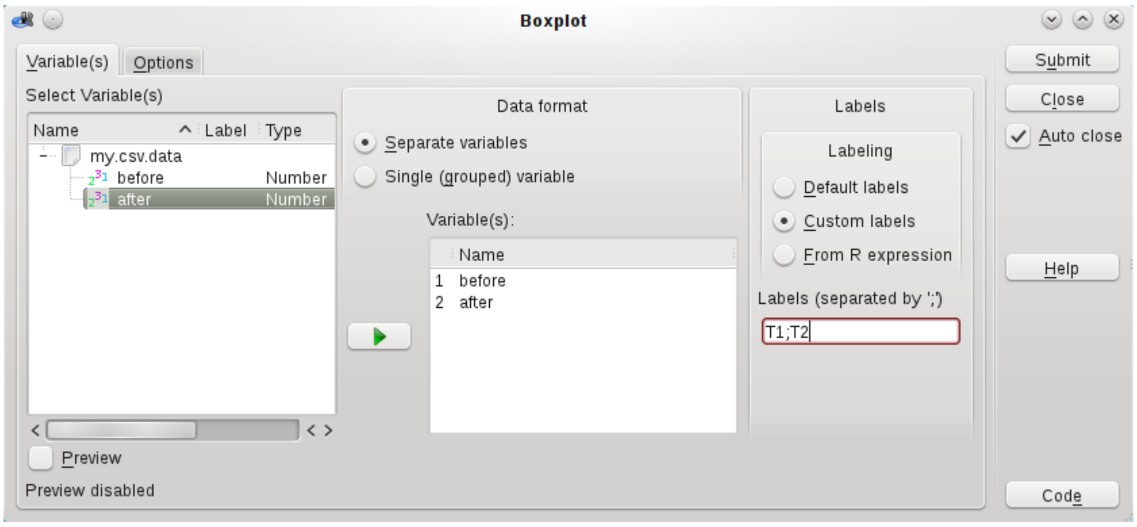


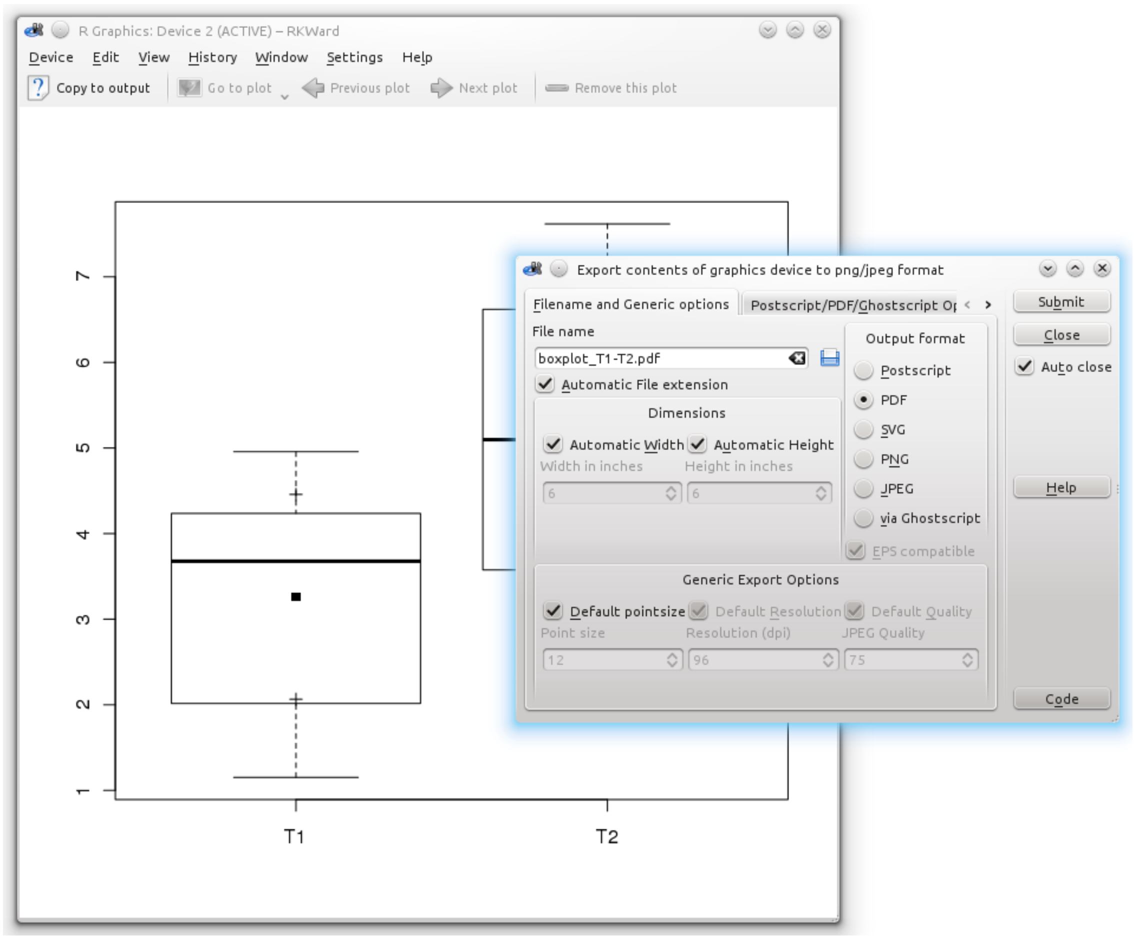
Figure 12: Boxplot dialog. The first tab `Variables` is used to select the variables for analysis. It is possible to combine any data present in `.GlobalEnv`. The second tab `Options` allows further adjustments (e.g., the addition of mean and standard deviation) to the plot (not shown).

5. Conclusion and outlook

In this article we have introduced the **RKWard** GUI to R. **RKWard** provides features ranging from easy to use dialogs for common statistical procedures targeted at R novices, to advanced IDE features targeted at R experts.

RKWard aims to empower users of all knowledge levels to make more efficient use of the R programming language, while carefully avoiding to lock in users to a specific GUI solution. In particular, **RKWard**

- Provides full transparency about the R code that is used to carry out tasks.
- Avoids introducing **RKWard**-specific R functions for central functionality (but uses some for output formatting).
- Avoids hard dependencies on third-party R packages.


R Graphics: Device 2 (ACTIVE) – RKWard
Device Edit View History Window Settings Help
Copy to output
Go to plot
Previous plot
Next plot
Remove this plot
1
2
3
4
5
6
7
T1
T2
Export contents of graphics device to png/jpeg format
Filename and Generic options
Postscript/PDF/Ghostscript Op
Submit
Close
Help
Code

File name: boxplot_T1-T2.pdf
[✓] Automatic File extension
Dimensions
[✓] Automatic Width [✓] Automatic Height
Width in inches: 6
Height in inches: 6
Output format: Postscript [✓ PDF] SVG PNG JPEG via Ghostscript
[✓] EPS compatible
Generic Export Options
[✓] Default pointsize [✓] Default Resolution [✓] Default Quality
Point size: 12
Resolution (dpi): 96
JPEG Quality: 75
[✓] Auto close

Figure 13: Plotted data and plot export dialog. The export dialog (`Device`→`Export`) provides numerous options like resolution and size for different vector formats (e.g., SVG, PDF) and pixel formats (e.g., PNG, JPEG). (Note: For the shown figure, the optional mean (■) and standard deviation (+) parameters were selected in the boxplot plugin.)

- Uses standard R formats (see R Development Core Team 2012b) for data storage, and open standards (HTML, PNG, SVG) for storage of output.

Future versions of **RKWard** will continue to add value for both groups of users. Planned features include an enhanced interface for debugging R code, support for editing more types of data, and the ability to connect the **RKWard** GUI to a remote R engine. Perhaps most importantly, **RKWard** will gain many new graphical dialogs for manipulation, analysis, and visualization of data. The ability to develop these dialogs as plugins allows to develop and distribute GUI dialogs independently of the **RKWard** core application, allowing any user to help in enhancing **RKWard**, without in-depth programming knowledge.

Acknowledgments

The software **RKWard**, presented in this paper, is currently developed by Thomas

Friedrichsmeier (lead developer), Prasenjit Kapat, Meik Michalke, and Stefan Rödiger. Many more people have contributed, or are still contributing to the project in various forms. We would like to thank (in alphabetical order) Adrien d'Hardemare, Daniele Medri, David Sibai, Detlef Steuer, Germán Márquez Mejía, Ilias Soumpasis, Jannis Vajen, Marco Martin, Philippe Grosjean, Pierre Ecochard, Ralf Tautenhahn, Roland Vollgraf, Roy Qu, Yves Jacolin, and many more people on `rkward-devel@lists.sourceforge.net` for their contributions.

The first two authors of this article have contributed equally, and both are available for correspondence pertaining to this article. Questions and comments regarding the software **RKWard** should be addressed to the project's main mailing list, `rkward-devel@lists.sourceforge.net`.

References

Blanchette J, Summerfield M (2008). *C++ GUI Programming with Qt 4*. 2nd edition. Prentice Hall.

Burnette E (2005). ***Eclipse** IDE Pocket Guide*. O'Reilly Media.

Cullmann C (2012). ***KatePart***. URL `http://kate-editor.org/about-katepart/`.

Dalgaard P (2001). "A Primer on the R-Tcl/Tk Package." *R News*, **1**(3), 27–31. URL `http://CRAN.R-project.org/doc/Rnews/`.

Dalgaard P (2002). "Changes to the R-Tcl/Tk Package." *R News*, **2**(3), 25–27. URL `http://CRAN.R-project.org/doc/Rnews/`.

Ettrich M, Taylor O (2002). *XEmbed Protocol Specification Version 0.5*. URL `http://standards.freedesktop.org/xembed-spec/xembed-spec-latest.html`.

Faure D (2000). "Creating and Using Components (KParts)." In D Sweet (ed.), ***KDE** 2.0 Development*. Sams, Indianapolis.

Fellows I (2012). "**Deducer**: A Data Analysis GUI for R." *Journal of Statistical Software*, **49**(8), 1–15. URL `http://www.jstatsoft.org/v49/i08/`.

Fox J (2005). "The R Commander: A Basic-Statistics Graphical User Interface to R." *Journal of Statistical Software*, **14**(9), 1–42. URL `http://www.jstatsoft.org/v14/i09/`.

Fox J (2007). "Extending the R Commander by "Plug-In" Packages." *R News*, **7**(3), 46–52. URL `http://CRAN.R-project.org/doc/Rnews/`.

Fox J (2008). "Editorial." *R News*, (2), 1–2. URL `http://CRAN.R-project.org/doc/Rnews/`.

Friedrichsmeier T, Michalke M (2011). *Introduction to Writing Plugins for **RKWard***. URL `http://rkward.sourceforge.net/documents/devel/plugins/index.html`.

Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JYH, Zhang

J (2004). "**Bioconductor**: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology*, **5**, R80. URL http://genomebiology.com/2004/5/10/R80.

Grosjean P (2010). "R GUI Projects." URL http://www.R-project.org/GUI.

Haumann D (2010). "**Kate**: Scripted Actions." URL http://kate-editor.org/2010/07/09/kate-scripted-actions/.

Helbig M, Urbanek S, Fellows I (2011). ***JGR**: Java Gui for R.* R Package Version 1.7-9, URL http://CRAN.R-project.org/package=JGR.

Hopkins D (2005). "HyperTIES Hypermedia Browser and Emacs Authoring Tool for NeWS." URL http://www.donhopkins.com/drupal/node/101.

Ihaka R (1998). "R: Past and Future History." In S Weisberg (ed.), *Proceedings of the 30th Symposium on the Interface*, pp. 392–396. The Interface Foundation of North America. URL http://CRAN.R-project.org/doc/html/interface98-paper/paper.html.

Jarvis S (2010). "**KDE** 4 on Windows." *Linux Journal*, **2010**.

KDE eV (2012). *About **KDE**.* Berlin. URL http://www.kde.org/community/whatiskde/.

KDevelopOrg (2011). ***KDevelop***. URL http://www.kdevelop.org/.

Kim J, Lutteroth C (2009). "Multi-Platform Document-Oriented GUIs." In G Weber, P Calder (eds.), *Tenth Australasian User Interface Conference (AUIC 2009)*, volume 93 of *CRPIT*, pp. 31–38. ACS, Wellington, New Zealand. URL http://crpit.com/confpapers/CRPITV93Kim.pdf.

KOfficeOrg (2010). ***KWord***. URL http://www.koffice.org/kword/.

Kuhn M (2006). "Sweave and the Open Document Format – The **odfWeave** Package." *R News*, **6**(4), 2–8. URL http://CRAN.R-project.org/doc/Rnews/.

Lecoutre E (2003). "The **R2HTML** Package – Formatting HTML Output on the Fly or by Using a Template Scheme." *R News*, **3**(3), 33–36. URL http://CRAN.R-project.org/doc/Rnews/.

Leisch F (2002). "Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *COMPSTAT 2002 – Proceedings in Computational Statistics*, pp. 575–580. Physica-Verlag, Heidelberg.

Ligges U (2003). "R Help Desk: Package Management." *R News*, **3**(3), 37–39. URL http://CRAN.R-project.org/doc/Rnews/.

Microsoft Developer Network (2010). "Microsoft Developer Network." URL http://msdn2.microsoft.com/en-us/library/ms997505.aspx.

Murdoch D (2002). "Reading Foreign Files." *R News*, **2**(1), 2–3. URL http://CRAN.R-project.org/doc/Rnews/.

Nokia Corporation (2012). ***Qt** – Cross-Platform Application and UI Framework.* Oslo. URL http://qt.nokia.com/.

Parikh A, Covington KR (2010). "**Red-R**: A Open Source Visual Programming GUI Interface for R." URL http://www.Red-R.org/.

Plate T (2009). ***trackObjs**: Track Objects.* R package version 0.8-6, URL http://CRAN.R-project.org/package=trackObjs.

Raaphorst S (2003). "A Usability Inspection of Several Graphical User Interface Toolkits." *Technical Report 51222*, University of Ottawa. URL http://www.cs.utoronto.ca/~sr/academic/csi51222paper.pdf.

Raffelsberger W, Krause Y, Moulinier L, Kieffer D, Morand AL, Brino L, Poch O (2008). "**RReportGenerator**: Automatic Reports from Routine Statistical Analysis Using R." *Bioinformatics*, **24**(2), 276–278.

R Development Core Team (2001). "What is R?" *R News*, (1), 2–3. URL http://CRAN.R-project.org/doc/Rnews/.

R Development Core Team (2012a). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

R Development Core Team (2012b). *R Data Import/Export.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-10-0, URL http://www.R-project.org/.

R Development Core Team (2012c). *R Internals.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-14-3.

R Development Core Team (2012d). *Writing R Extensions.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-11-9.

Ripley BD (2004). "Lazy Loading and Packages in R 2.0.0." *R News*, **4**(2), 2–4. URL http://CRAN.R-project.org/doc/Rnews/.

Ripley BD (2005). "Packages and Their Management in R 2.1.0." *R News*, **5**(1), 8–11. URL http://CRAN.R-project.org/doc/Rnews/.

Rödiger S, Ruhland M, Schmidt C, Schröder C, Grossmann K, Böhm A, Nitschke J, Berger I, Schimke I, Schierack P (2011). "Fluorescence Dye Adsorption Assay to Quantify Carboxyl Groups on the Surface of Poly(methyl methacrylate) Microbeads." *Analytical Chemistry*, **83**(9), 3379–3385.

Rödiger S, Schierack P, Böhm A, Nitschke J, Berger I, Frömmel U, Schmidt C, Ruhland M, Schimke I, Roggenbuck D, Lehmann W, Schröder C (2012). "A Highly Versatile Microscope Imaging Technology Platform for the Multiplex Real-Time Detection of Biomolecules and Autoimmune Antibodies." *Advances in Biochemical Bioengineering/Biotechnology.*

Rugg-Gunn PJ, Cox BJ, Ralston A, Rossant J (2010). "Distinct Histone Modifications in Stem Cell Lines and Tissue Lineages from the Early Mouse Embryo." *Proceedings of the National Academy of Sciences*, **107**(24), 10783–10790.

Sarkar D (2008). ***lattice**: Multivariate Data Visualization with R*. Springer-Verlag, New York. URL `http://lmdvr.R-Forge.R-project.org/`.

Snow G (2012). ***TeachingDemos**: Demonstrations for Teaching and Learning.* R package version 2.8, URL `http://CRAN.R-project.org/package=TeachingDemos`.

Sutherland WR (1966). *The On-line Graphical Specification of Computer Procedures*. Ph.D. thesis, MIT. URL `http://hdl.handle.net/1721.1/13474`.

Visne I, Dilaveroglu E, Vierlinger K, Lauss M, Yildiz A, Weinhaeusel A, Noehammer C, Leisch F, Kriegner A (2009). "**RGG**: A General GUI Framework for R Scripts." *BMC Bioinformatics*, **10**(1), 74.

Yang L, Liu J, Liu M, Qian M, Zhang M, Hu H (2011). "Identification of Fatty Acid Synthase from the Pacific White Shrimp, Litopenaeus vannamei and its Specific Expression Profiles During White Spot Syndrome Virus Infection." *Fish & Shellfish Immunology*, **30**(2), 744–749.

Zhang J, Gentleman R (2004). "Tools for Interactively Exploring R Packages." *R News*, **4**(1), 20–25. URL `http://CRAN.R-project.org/doc/Rnews/`.

Zou W, Tolstikov VV (2008). "Probing Genetic Algorithms for Feature Selection in Comprehensive Metabolic Profiling Approach." *Rapid Communications in Mass Spectrometry*, **22**(8), 1312–1324.

Zou W, Tolstikov VV (2009). "Pattern Recognition and Pathway Analysis with Genetic Algorithms in Mass Spectrometry Based Metabolomics." *Algorithms*, **2**(2), 638–666.

A. Appendix overview

In this appendix we will give an overview of some key aspects of **RKWard**'s technical design and development process, comparing them briefly to competing GUI solutions, where appropriate. We will give slightly more attention to the details of the plugin framework (Section F) used in **RKWard**, since this is central to the extensibility of **RKWard**, and we will conclude with an example for extending **RKWard** by a plugin (Section G).

Note that this document refers to **RKWard** version 0.5.6. Several technical details, described here, have changed in **RKWard** version 0.5.7 and the current development version.

B. Asynchronous command execution

One central design decision in the implementation of **RKWard** is that the interface to the R engine operates asynchronously. The intention is to keep the application usable to a high degree, even during the computation of time-consuming analysis. For instance, while waiting for the estimation of a complex model to complete, the user should be able to continue to use the GUI to prepare the next analysis. Asynchronous command execution is also a prerequisite for an implementation of the plot-preview feature (see Section 3.6). Internally, the GUI frontend and the R engine run in two separate processes[14]. Commands generated from plugins or user actions are placed in queue in the frontend and are evaluated in the backend process in the order they were submitted[15].

The asynchronous design implies that **RKWard** avoids relying on the R engine during interactive use. This is one of several reasons for the use of ECMAScript in plugins, instead of scripting using R itself (see Sections D and F). A further implication is that **RKWard** avoids querying information about the existence and properties of objects in R interactively. Rather, **RKWard** keeps a representation of R objects and their basic properties (e. g., class and dimensions), which is used for the workspace browser, object name completion, function argument hinting, and other places. The object representation includes objects in all environments in the search path, and any objects contained within these environments in a hierarchical tree[16]. The representation of R objects is gathered pro-actively[17]. This has a notable impact on performance when loading packages. Specifically, objects which would usually be "lazy loaded" only when needed (see Ripley 2004) are accessed in order to fetch information on their properties. This means the data has to be loaded from disk; however, the memory is freed immediately after fetching information on the object. Additionally, for packages with extremely large number of objects, **RKWard** provides an option to exclude specific packages from scanning the object structures.

A further side-effect of the asynchronous threaded design is that there is inherently a rather

[14] Up to **RKWard** version 0.5.4, two separate threads inside a single process were used. This alternate design is still available as a compile time option.

[15] It is possible, and in some cases necessary, to enforce a different order of command execution in internal code. For instance, **RKWard** makes sure that no user command can potentially interfere while **RKWard** is loading the data of a `data.frame` for editing.

[16] Currently, environments of functions or formulas are not taken into account, but slots of S4 objects, and package namespace environments are represented in the object tree.

[17] To limit the amount of processing, and to avoid recursion, **RKWard** currently stops gathering object information at a depth of three levels. Information on deeper levels is gathered on an as-needed basis, when the user accesses information on the respective parent objects.

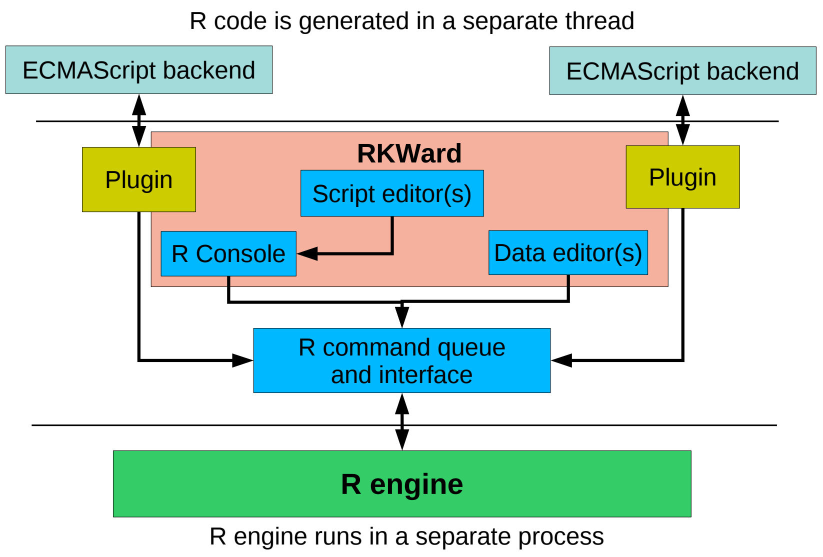

R code is generated in a separate thread
ECMAScript backend
ECMAScript backend
Plugin
RKWard
Script editor(s)
Plugin
R Console
Data editor(s)
R command queue and interface
R engine
R engine runs in a separate process

Figure 14: Technical design of **RKWard**. Only a few central components are visualized. All communication with the R engine is passed through a single interface living in the frontend process. The R engine itself runs in a separate process. Separate threads within the frontend process are used to generate R code from plugins.

clear separation between the GUI code and the code making direct use of the R application programming interface (API) (see also Figure 14). In future releases it could be made possible to run GUI and R engine on different computers.

C. Object modification detection

RKWard allows the user to run arbitrary commands in R at any time, even while editing a `data.frame` or while selecting objects for analysis in a GUI dialog. Any user command can potentially add, modify, or remove objects in R. **RKWard** tries to detect such changes in order to always display accurate information in the workspace browser, object selection lists, and object views. Beyond that, detecting any changes is particularly important with respect to objects which are currently being edited in the data editor (which provides an illusion of in-place editing, see Section 3.4). Here, it is necessary to synchronize the data between R and the GUI in both directions.

For simplicity and performance, object modification detection is only implemented for objects inside the "global environment" (including environments inside the global environment), since this is where changes are typically done. Currently, object modification detection is based on active bindings. Essentially, any object which is created in the global environment is first moved to a hidden storage environment, and then replaced with an active binding. The active binding acts as a transparent proxy to the object in the storage environment, which registers any write-access to the object[18].

[18] This is similar to the approach taken in the **trackObjs** package (Plate 2009).

The use of active bindings has significant performance implications when objects are accessed very frequently. This is particularly notable where an object inside the global environment is used as the index variable in a loop, as illustrated by the following example. When control returns to the top level prompt, after the first assignment, `i` will become subject to object modification detection (i. e., it will be wrapped into an active binding). The subsequent `for` loop will then run slow.

```
R> i <- 1
R> for (i in 1:100000) i + i
```

In contrast, in the following example, `i` is a local object, and will not be replaced by an active binding. Therefore the loop will run approximately as fast as in a plain R session:

```
R> f <- function () {
+    i <- 1
+    for (i in 1:100000) i + i
+  }
R> f ()
```

Future versions of **RKWard** will try to avoid this performance problem. One approach that is currently under consideration is to simply perform a pointer comparison of the `SEXP` records of objects in global environment with their copies in a hidden storage environment. Due to the implicit sharing of `SEXP` records (R Development Core Team 2012d,c), this should provide for a reliable way to detect changes for most types of R objects, with comparatively low memory and performance overhead. Special handling will be needed for environments and active bindings.

D. Choice of toolkit and implementation languages

In addition to R, **RKWard** is based on the **KDE** libraries (**KDE** e.V. 2012), which are in turn based on **Qt** (Nokia Corporation 2012), and implemented mostly in C++. Compared to many competing libraries, this constitutes a rather heavy dependency. Moreover, the **KDE** libraries are still known to have portability issues especially on Mac OS X, and to some degree also on the Microsoft Windows platform (Jarvis 2010).

The major reason for choosing the **KDE** and **Qt** libraries has been the many high level features, they provide. This has allowed **RKWard** development to make quick progress despite limited resources. Most importantly, the **KDE** libraries provide a full featured text editor (Cullmann 2012) as a component which can be seamlessly integrated into a host application using the KParts technology (Faure 2000). Additionally, another KPart provides HTML browsing capabilities in a similarly integrated way. The availability of **KWord** (KOffice.Org 2010) as an embeddable KPart might prove useful in future versions of **RKWard**, when better integration with office suites will be sought. Additionally **Qt** libraries offer the encapsulation of the look-and-feel on specific platforms for a high degree of interoperability and a wide selection of powerful widgets (Raaphorst 2003).

Another technology from the **KDE** libraries that is important to the development of **RKWard** is the "XMLGUI" technology (Faure 2000). This is especially helpful in providing an integrated GUI across the many different kinds of document windows and tool views supported in **RKWard**.

Plugins in **RKWard** rely on XML (`http://www.w3.org/XML/`) and ECMAScript (`http://www.ecmascript.org/`; see Section F). XML is not only well suited to describe the layout of the GUI of plugins, but simple functional logic can also be represented (see also Visne *et al.* 2009). ECMAScript was chosen for the generation of R commands within plugins, in particular due to its availability as an embedded scripting engine inside the **Qt** libraries. While at first glance R itself would appear as a natural choice of scripting language as well, this would make it impossible to use plugins in an asynchronous way. Further, the main functional requirement in this place is the manipulation and concatenation of text strings. While R provides support for this, concatenating strings with the `+` operator, as available in ECMAScript, allows for a very readable way to perform such basic text manipulation.

E. On-screen graphics windows

Contrary to the approach used in **JGR** (Helbig, Urbanek, and Fellows 2011), **RKWard** does not technically provide a custom on-screen graphics device. **RKWard** detects when new graphics windows are created via calls to `X11()` or `windows()`. These windows are then "captured" in a platform dependent way (based on the XEmbed (Ettrich and Taylor 2002) protocol for **X11**, or on reparenting for the Microsoft Windows platform). An **RKWard** menu bar and a toolbar is then added to these windows to provide added functionality. While this approach requires some platform dependent code, any corrections or improvements made to the underlying R native devices will automatically be available in **RKWard**.

A recent addition to the on-screen device is the "plot history" feature which adds a browsable list of plots to the device window. Since **RKWard** does not use a custom on-screen graphics device, this feature is implemented in a package dependent way. For example, as of this writing, plotting calls that use either the "standard graphics system" or the "**lattice** system" can be added to the plot history; other plots are drawn but not added. The basic procedure is to identify changes to the on-screen canvas and record the existing plot before a new plot wipes it out. A single global history for the recorded plots is maintained which is used by all the on-screen device windows. This is similar to the implementation in Rgui.exe (Microsoft Windows), but unlike the one in Rgui.app (Mac OS X). Each such device window points to a position in the history and behaves independently when recording a new plot or deleting an existing one.

Plot history support for the **lattice** system (Sarkar 2008) is implemented by inserting a hook in the `print.lattice()` function. This hook retrieves and stores the `lattice.status` object from the `lattice:::.LatticeEnv` environment, thereby making `update()` calls on trellis objects transparent to the user. Any recorded trellis object is then replayed using `plot.lattice()`, bypassing the recording mechanism. The standard graphics system, on the other hand, is implemented differently because the hook in `plot.new()` is ineffective for this purpose. A customized function is overloaded on `plot.new()` which stores and retrieves the existing plot, essentially, using `recordPlot()` and replays them using `replayPlot()`.

The actual plotting calls are tracked using appropriate `sys.call()` commands in the hooks. These call strings are displayed as a drop-down menu on the toolbar for non-sequential browsing (see Section 3.6) providing a very intuitive browsing interface unlike the native implementations in `windows()` and `quartz()` devices.

F. Plugin infrastructure

One of the earliest features of **RKWard** was the extensibility by plugins. Basically, plugins in **RKWard** provide complete GUI dialogs, or re-usable GUI components, which accept user settings and translate those user settings into R code[19]. Thus, the plugin framework is basically a tool set used to define GUIs for the automatic generation of R code.

Much of the functionality in **RKWard** is currently implemented as plugins. For example, importing different file formats relying on the **foreign** package is achieved by this approach. Similarly, **RKWard** provides a modest GUI driven tool set for statistical analysis, especially for item response theory, distributions, and descriptive statistical analysis.

F.1. Defining a plugin

Plugins consist of four parts as visualized in Figure 15 (see Section G for an example; for a complete manual, see Friedrichsmeier and Michalke 2011):

- An XML file (Section G.1), called a "plugin map", is used to declare one or more plugins, each with a unique identifier. For most plugins, the plugin map also defines the placement in the menu hierarchy. Plugin maps are meant to represent groups of plugins. Users can disable/enable such groups of plugins in order to reduce the complexity of the menu hierarchy.

- A second XML file describes the plugin GUI layout itself (Section G.2). Most importantly this includes the definition of the GUI layout and GUI behavior. High level GUI elements can be defined with simple XML-tags. Layout is based on "rows" and "columns", instead of pixel counts. In most cases this allows for a very sensible resizing behavior. **RKWard** supports single-page dialogs and multi-page wizards, however, most plugins define only a single-page GUI. GUI behavior can be programmed by connecting "properties" of the GUI elements to each other. For example, the state of a checkbox could be connected to the "enabled" property of a dependent control. More complex logic is also supported, as is procedural scripting of GUI behavior using ECMAScript.

- A separate ECMAScript file (Section G.3) is used to translate GUI settings into R code[20]. This ECMAScript file is evaluated asynchronously in a separate thread. **RKWard** currently enforces structuring the code into three separate sections for preprocessing, calculating, and printing results. The generated code is always run in a local environment, in order to allow the use of temporary variables without the danger of overwriting user data.

- A third XML file defines a help page. This help page usually links to the R help pages of the main functions/concepts used by the plugin, as well as to other related **RKWard** help pages. Compared to R help pages, the plugin help pages try to give more hands-on

[19] Plugins are also used in some other contexts within **RKWard**, for instance, the integrated text editor (**kate** part) supports extensions via plugins and user scripts. At this point we will focus only on plugins generating R code.

[20] In earlier versions of **RKWard**, PHP was used as a scripting engine, and PHP interpreters were run as separate processes. Usage of PHP was abandoned in **RKWard** version 0.5.3 for reasons of performance and simplicity.

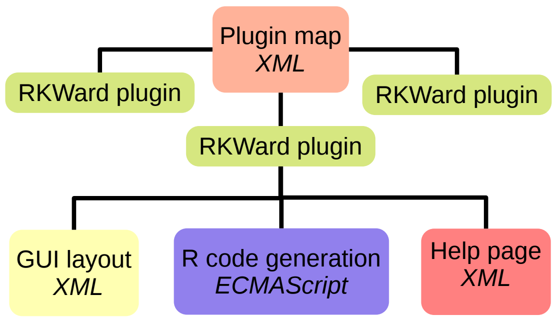


Figure 15: Plugin structure of **RKWard**. One or more plugins are declared in a "plugin map". Each plugin is defined by two XML files, and one ECMAScript file.

advice on using the plugin. Plugins can be invoked from their help page by clicking on a link near the top, which can be useful after following a link from a related help page.

Changes to the source code of these elements take effect without the requirement to recompile **RKWard**.

F.2. Embedding and reuse of plugins

RKWard supports several mechanisms for modularization and re-use of functionality in plugins. File inclusion is one very simple but effective mechanism, which can be used in the ECMAScript files, but is also supported in the XML files. In script files, this is most useful by defining common functions in an included file. For the XML files, the equivalent is to define "snippets" in the included file, which can then be inserted.

A third mechanism allows to completely embed one plugin into another. For instance the `plot_options` plugin is used by many plugins in **RKWard**, to provide common plot options such as labels, axis options, and grids. Other plugins can embed it using the `embed`-tag in their XML file (the plugin supports hiding irrelevant options). The generated code portions can be fetched from the ECMAScript file just like any other GUI settings, and inserted into the complete code. Other examples of embedded plugins are options for histograms, barplots, and empirical cumulative distribution function (ECDF) plots (which in turn embed the generic plot options plugin).

F.3. Enforcing a consistent interface

RKWard tries to make it easy to create a consistent interface in all plugins. GUI-wise this is supported by providing high-level GUI elements, and embeddable clients. Also, the standard elements of each dialog (`Submit`, and `Cancel` buttons, on-the-fly code view, etc.) are hard coded. Up to version 0.5.3 of **RKWard** it was not possible to use any GUI elements in plugins which were not explicitly defined for this purpose. In the current development version, theoretically, all GUI elements available from **Qt** can be inserted, where necessary.

For generating output, the function `rk.header()` can be used to print a standardized caption for each piece of output. Printing results in vector or tabular form is facilitated by `rk.results()`. A wide range of objects can be printed using `rk.print()`, which is just a thin wrapper around the `HTML()` function of the **R2HTML** package (Lecoutre 2003) in the current implementation. The use of custom formatting with HTML is possible, but discouraged. Standard elements such as a horizontal separator, and the `Run again` link (see Section 3.7) are inserted automatically, without the need to define them for each plugin.

Regarding the style of the generated R code, enforcing consistency is harder, but plugins which are to become part of the official **RKWard** application are reviewed for adherence to some guidelines. Perhaps the most important guidelines are

- Write readable code, which is properly indented, and commented where necessary.
- Do not hide any relevant computations from the user by performing them in the ECMAScript. Rather, generate R code which will perform those computations, transparently.
- Plugins can be restricted to accept only certain types of data (such as only one-dimensional numeric data). Use such restrictions where appropriate to avoid errors, but be very careful not to add too many of them.

F.4. Handling of R package dependencies

A wide range of plugins for diverse functionality is present in **RKWard**, including plots (e. g., boxplot) or standard tests (e. g., Student's *t* test)[21]. Some of the plugins depend on R packages other than the recommended R base packages. Examples herein are the calculation of kurtosis, skewness or the exact Wilcoxon test.

RKWard avoids loading all these packages pro-actively, as **Rcmdr** does. Rather, plugins which depend on a certain package simply include an appropriate call to `require()` in the pre-processing section of the generated R code. The `require()` function is overloaded in **RKWard**, in order to bring up the package-installation dialog whenever needed. Packages invoked by `require()` remain loaded in the active **RKWard** session unless unloaded manually (from the workspace browser, or using the R function `detach()`).

Dependencies between (embedded) plugins are handled using the `<require>`-tag in the plugin map.

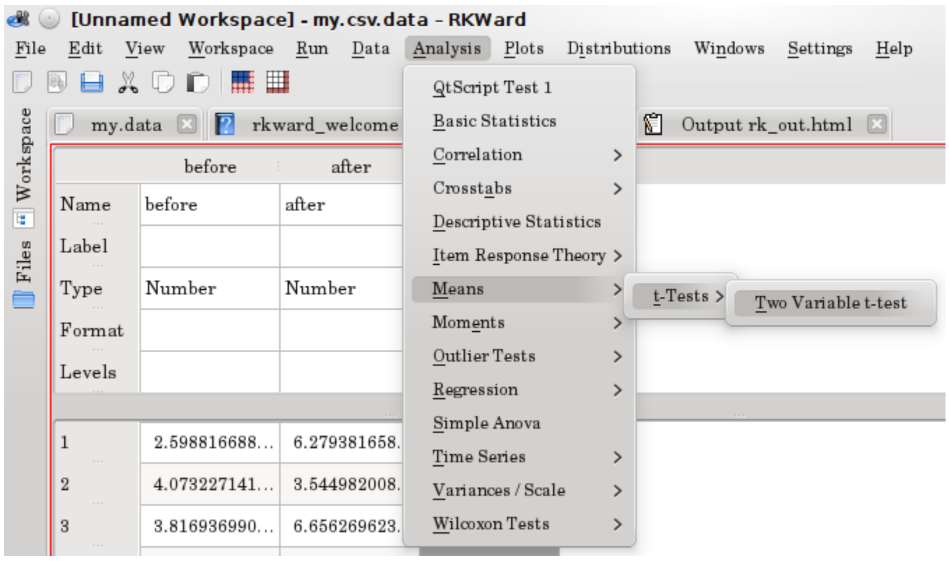
F.5. Development process

RKWard core and external plugins

Newly developed plugins are placed in a dedicated plugin map file[22]. Plugins in this map are not visible to the user by default, but need to be enabled manually. Once the author(s) of a plugin announces that they consider it stable, the plugin is subjected to a review for correctness, style, and usability. The review status is tracked in the project wiki. Currently at least one positive review is needed before the plugin is allowed to be made visible by default, by moving it to an appropriate plugin map.

[21] At the time of this writing, there are 164 user-accessible plugins in **RKWard**. Listing all is beyond the scope of this article.

[22] `under_development.pluginmap`



	before	after
Name	before	after
Label		
Type	Number	Number
Format		
Levels		
1	2.598816688...	6.279381658.
2	4.073227141...	3.544982008.
3	3.816936990...	6.656269623.

Figure 16: Generated menu structure as defined by the plugin map.

With the release of version 0.5.5, **RKWard** gained support for downloading additional sets of plugins directly from the internet. By simply clicking an `Install` button in a graphical dialog (`Settings`→`Configure RKWard`→`Plugins`), an external plugin set is downloaded, unpacked and its plugin map added to **RKWard**'s configuration, so it becomes instantly available after the configuration dialog is closed. External plugin sets are neither officially included nor supported by the **RKWard** developers. However, they allow plugin developers to easily extend **RKWard** with state-of-the-art or highly specialized features. To achieve this, **RKWard** (version 0.5.6) draws on **KNewStuff2**, a **KDE** library providing support for **GHNS**[23].

Automated testing

A second requirement for new plugins is that each plugin must be accompanied by at least one automated test. The automated testing framework in **RKWard** consists of an R package, **rkwardtests**, providing a set of R functions which allow to run a plugin with specific GUI settings, automatically. The resulting R code, R messages, and output are then compared to a defined standard. Automated tests are run routinely after changes in the plugin infrastructure, and before any new release.

The automated testing framework is also useful in testing some aspects of the application which are not implemented as plugins, but this is currently limited to very few basic tests.

G. Extending RKWard: An example for creating a plugin

As discussed in Section F, plugins in **RKWard** are defined by four separate files (Figure 15). To give an impression of the technique, this section shows (portions of) the relevant files for a plugin that provides a simple dialog for a Student's t test. For brevity, the help-file is omitted.

[23] **GHNS** (Get Hot New Stuff) is a technology platform (software and specifications) for desktop users to share their work. It is hosted under the umbrella of the freedesktop.org project at `http://ghns.freedesktop.org`. In future versions of **RKWard**, this framework will be deprecated in favor of standard R packages.

G.1. Defining the menu hierarchy

A so called `.pluginmap` file declares each plugin, and, if appropriate, defines where it should be placed in the menu hierarchy. Usually each `.pluginmap` file declares many plugins. In this example we only show one, namely, a two variable Student's t test (see Figure 16). The pluginmap (`<!DOCTYPE rkpluginmap>`) gives a unique identifier ("id"), the location of the GUI description ("file"), and the window title ("label"). The menu layout is defined in a hierarchical structure by nesting `<menu>` elements to form toplevel menus and submenus. Menus with the same "id" are merged across `.pluginmap` files. Moreover, the position within the menu can be explicitly defined (attribute "index"). This might be required if the menu entries are to be ordered non-alphabetically.

```
<!DOCTYPE rkpluginmap>
<document base_prefix="" namespace="rkward">
  <components>
    <component type="standard" id="t_test_two_vars"
          file="demo_t_test_two_vars.xml" label="Two Variable t-test" />
  </components>
  <hierarchy>
    <menu id="analysis" label="Analysis" index="4">
      <menu id="means" label="Means" index="4">
        <menu id="ttests" label="t-Tests">
          <entry component="t_test_two_vars" />
        </menu>
      </menu>
    </menu>
  </hierarchy>
</document>
```

G.2. Defining the dialog GUI

The main XML file of each plugin defines the layout and behavior of the GUI, and references the ECMAScript file that is used for generating R code from GUI settings and the help file (not included in this paper). GUI logic can be defined directly in the XML file (the `<logic>` element). In this example, the `Assume equal variances` checkbox is only enabled for paired sample tests. Optionally, GUI behavior can also be scripted in ECMAScript.

The XML file defines the Student's t test plugin (`<!DOCTYPE rkplugin>`) to be organized in two tabs[24]. On the first tab, two variables can be selected (`<varslot .../>`). These are set to be `required`, i. e., the `Submit` button will remain disabled until the user has made a valid selection for both. The second tab includes some additional settings like the confidence level (default 0.95).

```
<!DOCTYPE rkplugin>
<document>
  <code file="demo_t_test_two_vars.js"/>
  <help file="demo_t_test_two_vars.rkh"/>

  <logic>
    <connect client="varequal.enabled" governor="paired.not"/>
  </logic>
```

[24]A screenshot of the resulting dialog can be found in Figure 10.

```
  <dialog label="Two Variable t-Test">
    <tabbook>
      <tab label="Basic settings" id="tab_variables">
        <row id="basic_settings_row">
          <varselector id="vars"/>
          <column>
            <varslot type="numeric" id="x" source="vars" required="true"
              label="compare"/>
            <varslot type="numeric" id="y" source="vars" required="true"
              label="against"/>
            <radio id="hypothesis" label="using test hypothesis">
              <option value="two.sided" label="Two-sided"/>
              <option value="greater" label="First is greater"/>
              <option value="less" label="Second is greater"/>
            </radio>
            <checkbox id="paired" label="Paired sample" value="1" value_unchecked="0" />
          </column>
        </row>
      </tab>
      <tab label="Options" id="tab_options">
        <checkbox id="varequal" label="assume equal variances" value="1"
          value_unchecked="0"/>
        <frame label="Confidence Interval" id="confint_frame">
          <spinbox type="real" id="conflevel" label="confidence level" min="0" max="1"
            initial="0.95"/>
          <checkbox id="confint" label="print confidence interval" value="1"
            checked="true"/>
        </frame>
        <stretch/>
      </tab>
    </tabbook>
  </dialog>
</document>
```

G.3. Generating R code from GUI settings

A simple ECMAScript script is used to generate R code from GUI settings (using `echo()` commands). Generated code for each plugin is divided into three sections: “Preprocess”, “Calculate”, and “Printout”, although each may be empty.

```
var x;
var y;
var varequal;
var paired;

function preprocess () {
  x = getValue ("x");
  y = getValue ("y");

  echo ('names <- rk.get.description (' + x + ", " + y + ')\n');
}

function calculate () {
  varequal = getValue ("varequal");
  paired = getValue ("paired");
  var conflevel = getValue ("conflevel");
  var hypothesis = getValue ("hypothesis");

  var options = ", alternative=\"" + hypothesis + "\"";
  if (paired) options += ", paired=TRUE";
  if ((!paired) && varequal) options += ", var.equal=TRUE";
  if (conflevel != "0.95") options += ", conf.level=" + conflevel;

  echo ('result <- t.test (' + x + ", " + y + options + ')\n');
}

function printout () {
  echo ('rk.header (result\$method, \n');
  echo ('  parameters=list ("Comparing", paste (names[1], "against", names[2]),\n');
  echo ('  "H1", rk.describe.alternative (result)');
  if (!paired) {
    echo (',\n');
    echo ('  "Equal variances", "');
    if (!varequal) echo ("not");
    echo (' assumed"');
  }
  echo ('))\n');
  echo ('\n');
  echo ('rk.results (list (\n');
  echo ('  \'Variable Name\'=names,\n');
  echo ('  \'estimated mean\'=result\$estimate,\n');
  echo ('  \'degrees of freedom\'=result\$parameter,\n');
  echo ('  t=result\$statistic,\n');
  echo ('  p=result\$p.value');
  if (getValue ("confint")) {
    echo (',\n');
    echo ('  \'confidence interval percent\'=(100 * attr(result\$conf.int, "conf.level")),\n');
    echo ('  \'confidence interval of difference\'=result\$conf.int ');
  }
  echo ('))\n');
}
```

Affiliation:

Stefan Rödiger
Lausitz University of Applied Sciences
Department of Bio-, Chemistry and Process Engineering
and
Kardiologie-CCM, Charité-Universitätsmedizin Berlin
Germany
E-mail: stefan_roediger@gmx.de, rkward-devel@lists.sourceforge.net

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 49, Issue 9 *Submitted:* 2010-12-28
June 2012 *Accepted:* 2011-05-06