Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

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

Stefan Rödiger[‡,§]
Lausitz University of Applied Sciences & Charité

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 cross-platform 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.

[‡]Equal contribution, [§]Corresponding authors

Keywords: cross-platform, 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 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 Tlc/Tk (**??**). Meanwhile a plethora of R GUIs have emerged (see http://www.sciviews.org/_rgui/ for a comprehensive list). In 2005 John Fox released version 1.0 of R Commander (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 (**??**). In November 2002 Thomas Friedrichsmeier started the **RKWard** open-source software project with the goal to create a GUI for R based on **KDE**[1] and **Qt**[2] technologies.

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[3]. 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** or **TeachingDemos**), but as a productive tool. Since its incarnation **RKWard** has gained acceptance for usage in peer-rewiewed publications (**?????**). Dialogs for statistical procedures in **RKWard** do not necessarily show a one-to-one correspon-

[1] http://www.kde.org/

[2] http://qt.nokia.com/

[3] This distinguishes **RKWard** from R GUIs such as **Red-R** (http://www.red-r.org/), which specifically aims to hide the complexities of the R programming language, following the concept of visual data-flow programming (**?**). In contrast, **RKWard** limits itself to generate R code from GUI settings.

dence 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 easily (see Sections 5.1 and 6).

RKWard is licensed under the terms of the GNU General Public License[4] 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[5]. 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 first give an overview over the main GUI elements and features of **RKWard** (Section 3), followed by a short example of a simple **RKWard** session (Section 4). Next, technical details of the implementation will be discussed, comparing them briefly to competing GUI solutions, where appropriate (Section 5). Finally, we show an example for creating a plugin extension to **RKWard** (Section 6).

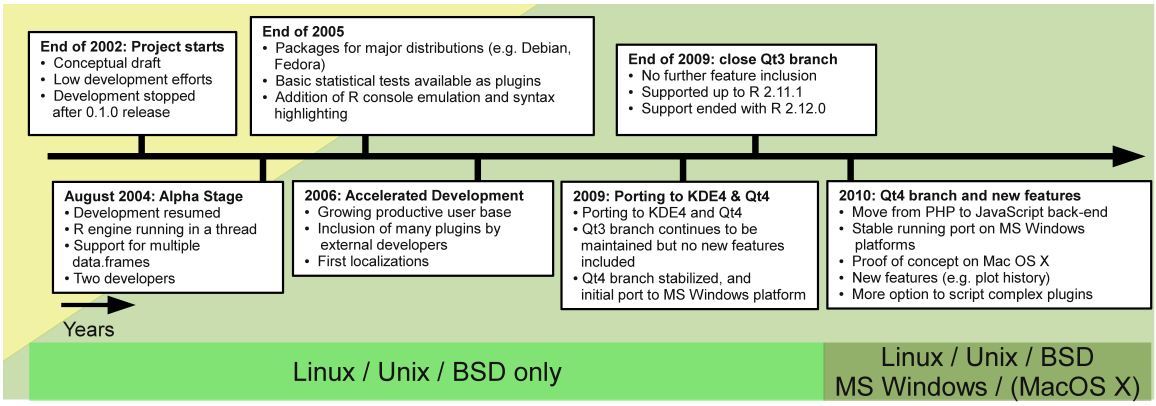

Figure 1: Timeline of important development milestones and changes in **RKWard**. Time is presented on an arbitrary scale. Here **Qt**3 and **Qt**4 refers to the 3.x and 4.x versions of the **Qt** libraries, respectively and **KDE**4 refers to the 4.x version of the **KDE** libraries.

[4] `http://www.gnu.org/licenses/gpl.html`
[5] `http://www.gnu.org/licenses/fdl.html`

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[6]. 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** originates on the GNU/Linux platform, and binary packages are available for many major distributions, including Debian, Ubuntu, OpenSuse, Gentoo, Fedora, and also FreeBSD. On systems which provide up-to-date packages of R and **KDE**, compilation from source is generally unproblematic[7]. The exact size of the installation is system dependent. On Debian x86, the package is currently around 1.5 MB (Megabyte) compressed, and 5.5MB uncompressed. However, if the **KDE** runtime environment is not yet installed, an installation of **RKWard** may need several hundred MB of disc space.

2.2. Installation on Microsoft Windows

RKWard can be used on Windows XP, Windows Server 2003, Vista, and Windows 7. Source installation on the Microsoft Windows platform is comparatively difficult, since various tools need to be installed[8], however, 32bit binaries are also provided by the project. The user has a choice between an small installer (1.7 MB), which can be used to install **RKWard** to pre-existing installations of R and **KDE**, and a installation bundle, which includes **RKWard**, R, and **KDE**. This bundle can be unpacked to any user-writable folder, and can be run without any further steps of installation. Using this method of installation, **RKWard** can also be installed to a removable storage medium, and moved between different systems (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.

2.3. Installation on Mac OS X

At the time of this writing, the developers lack the resources to support a Mac OS X port, and especially to provide binaries for Mac OS X. **RKWard** has been successfully compiled and installed on the Mac, and appeared to be mostly functional, but there have also been unresolved reports of failure to compile or to start **RKWard** on Mac OS X. Since **KDE** project does not currently offer binaries for Mac OS X, installation of **RKWard** also requires compilation of the **KDE** runtime environment, and its dependencies from source, which takes many hours to complete on current systems. Further, **RKWard**'s grapics device window related features (see Section 3.6) are only available when compiling and using **KDE** and

[6] See http://p.sf.net/rkward/download for an overview and platform specific download links.
[7] See http://p.sf.net/rkward/compilling for details.
[8] See http://sourceforge.net/apps/mediawiki/rkward/index.php?title=RKWard$_o n_W indows/Packaging for details$.

RKWard in **X11** mode. In conclusion, **RKWard** on Mac OS X cannot currently be considered ready for regular users.

2.4. Starting RKWard

RKWard cannot be loaded from within an R session, but rather it is 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 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[9] 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.

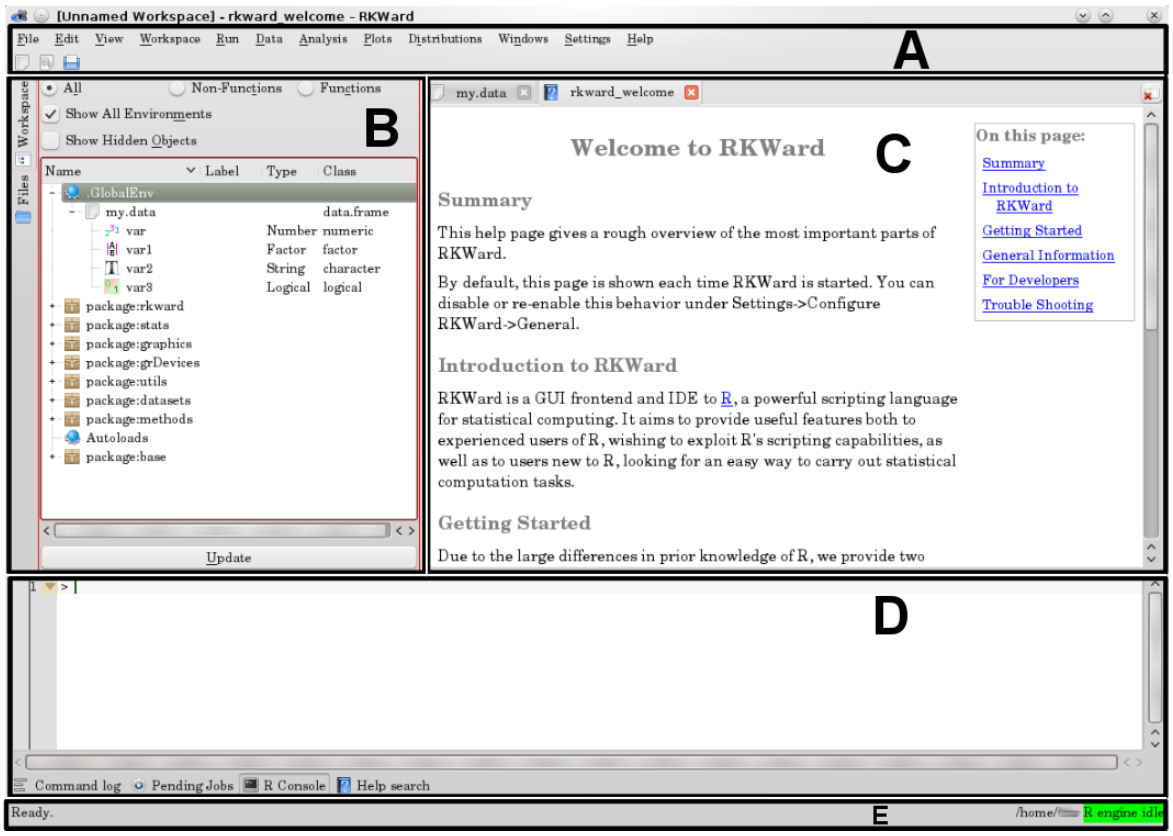

Figure 2: Default **RKWard** main window after start up. A) Menu bar and toolbar, B) tool panel showing workspace browser, C) document view area, showing a help page, D) tool panel showing embedded R console, and E) status bar. Panels B and D can be resized or collapsed. The red border around B indicates that the workspace browser is the active interface element.

A status bar is shown at the bottom of the window (Figure 2E). It displays (from right to left) 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

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

hovering the mouse pointer over them.

The central area is divided into a document view area (Figure 2C) and two panel subwindows (Figure 2B and D). The panels can be resized or moved to another edge of the central area independently. All panels can be toggled by mouse or keyboard shortcuts. When a panel is closed, the document view area (see below) is automatically re-sized to take up the free space.

The left panel (Figure 2B) contains a file browser (see Section 3.9) and a workspace browser (see Section 3.1) by default. The bottom panel (Figure 2D) contains the tool windows, namely, Command log (Section 3.9), Pending Jobs (Section 3.9), R Console (Section 3.3), and Help Search (Section 3.10).

The remainder of the central area (Figure 2C) is a single row tabbed document interface (TDI) for different documents. Early uses of TDIs date back to 1988 and are widely applied nowadays (**???**). Currently, the supported types of documents are 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 **??**). The order of tabs can be conveniently re-arranged using drag & drop.

All document windows and tool views can be detached from and re-attached to the main window as independent windows, managed by the window manager. This feature allows to conveniently work with multiple documents at the same time, e. g., scripts or data editors. 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 shown (or toggled) using a mouse device with point & click, as well as using a series of keyboard shortcuts (defined by default) for switching between the different tool and document windows. 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** 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[10]. 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

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

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. The editor automatically 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[11]. 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, the "File" menu, or by creating a new empty script. 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 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.

[11] 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.

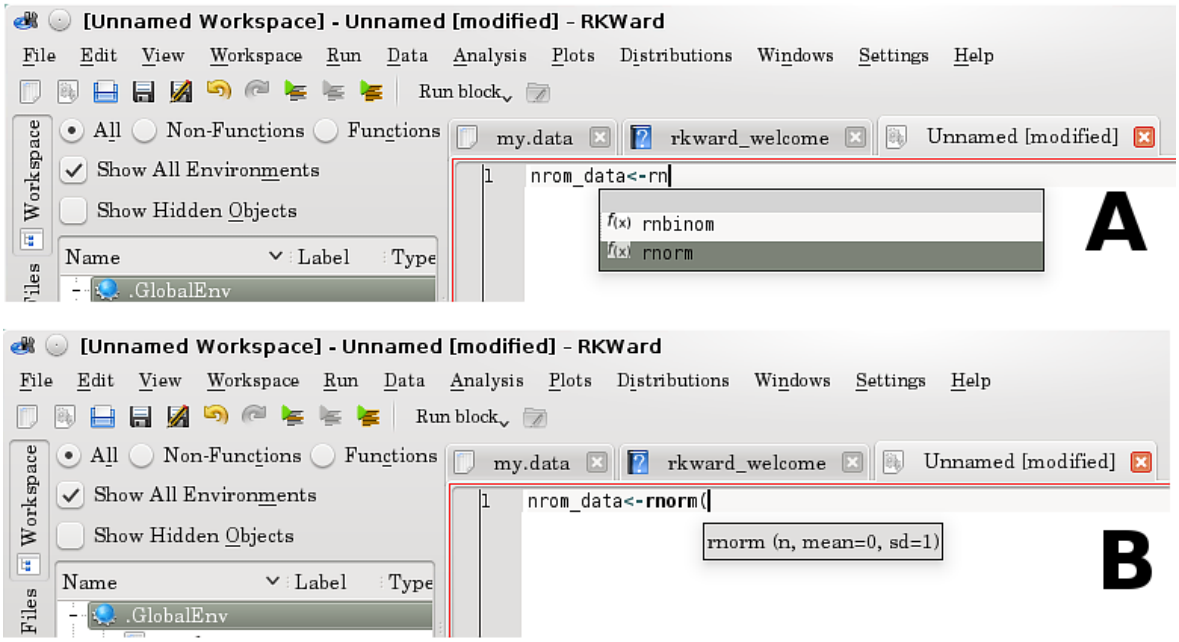

Figure 3: Code hinting features of the script editor. The script editor is able to hint A) R object names and B) function arguments.

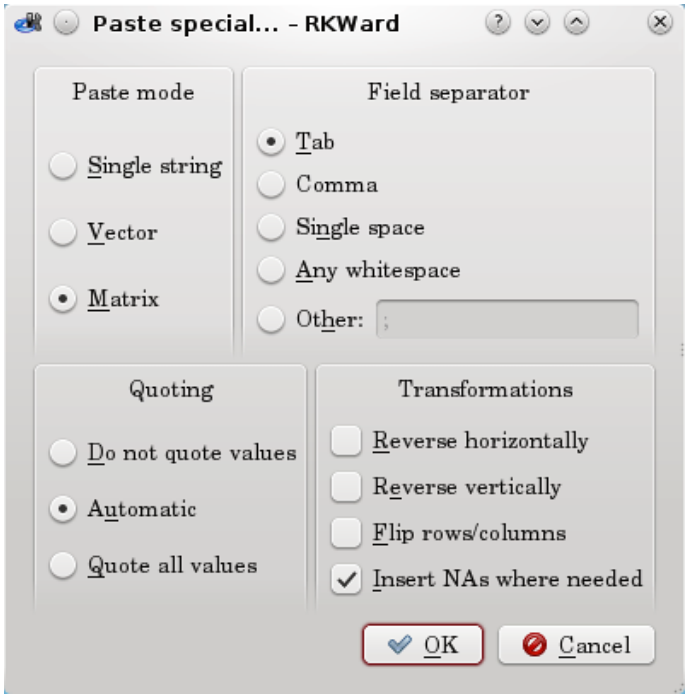

Figure 4: Paste special dialog. This tool allows to paste data (e. g., tabular, text) from the clipboard, directly to an R script and therefore accelerates the work process with data from different sources like spreadsheet applications.

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 the submitted in the background, instead. For further technical details, see Section 5.

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

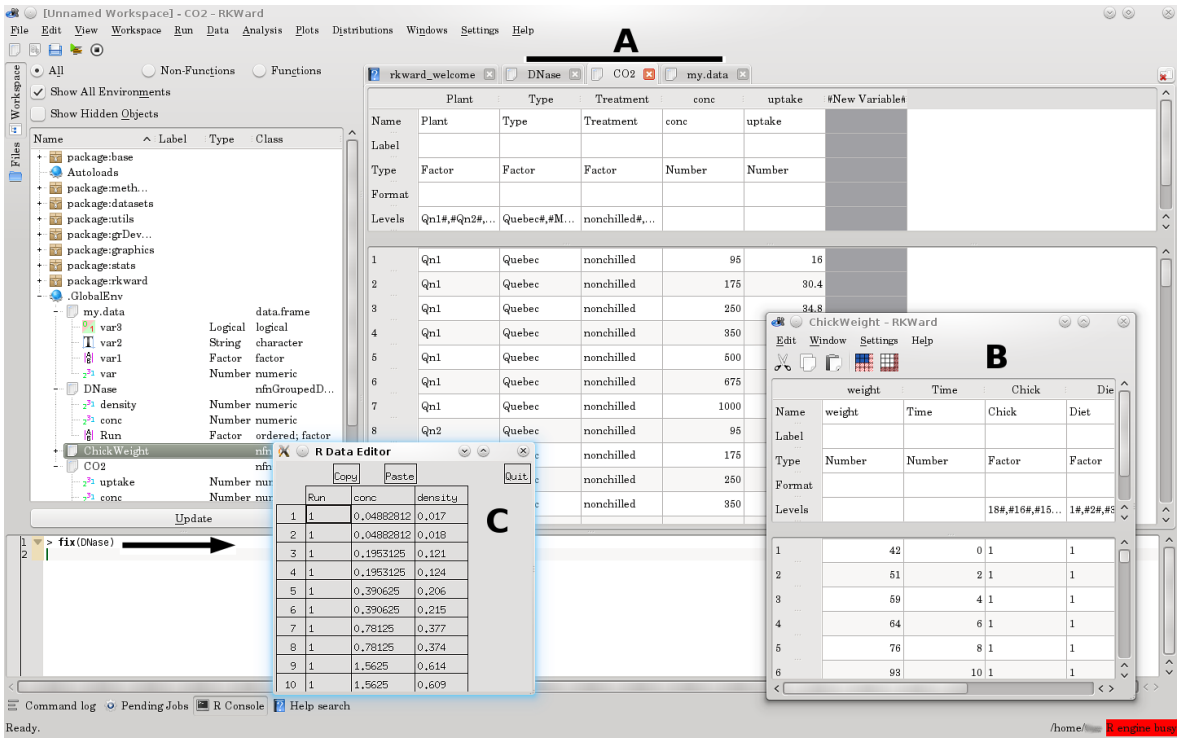

Figure 5: **RKWard** with several `data.frame`s in use at the same time. A) One `data.frame` is opened for editing in the main window. Two further `data.frame`s are opened in the background in tabs. B) Another `data.frame` is opened as detached window. C) R's standard data editing features (e. g., `fix()`,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

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**. 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 Section 5.1 for details). 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[12]. Further examples are given in Section 4.

At the time of this writing, **RKWard** provides support for importing SPSS, Stata, and "delimited text" data. Internally, **RKWard** relies on standard R functions and the package **foreign** (**?**) for reading these data files. To import CSV data, select "File→Import format→Import

[12] 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).

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

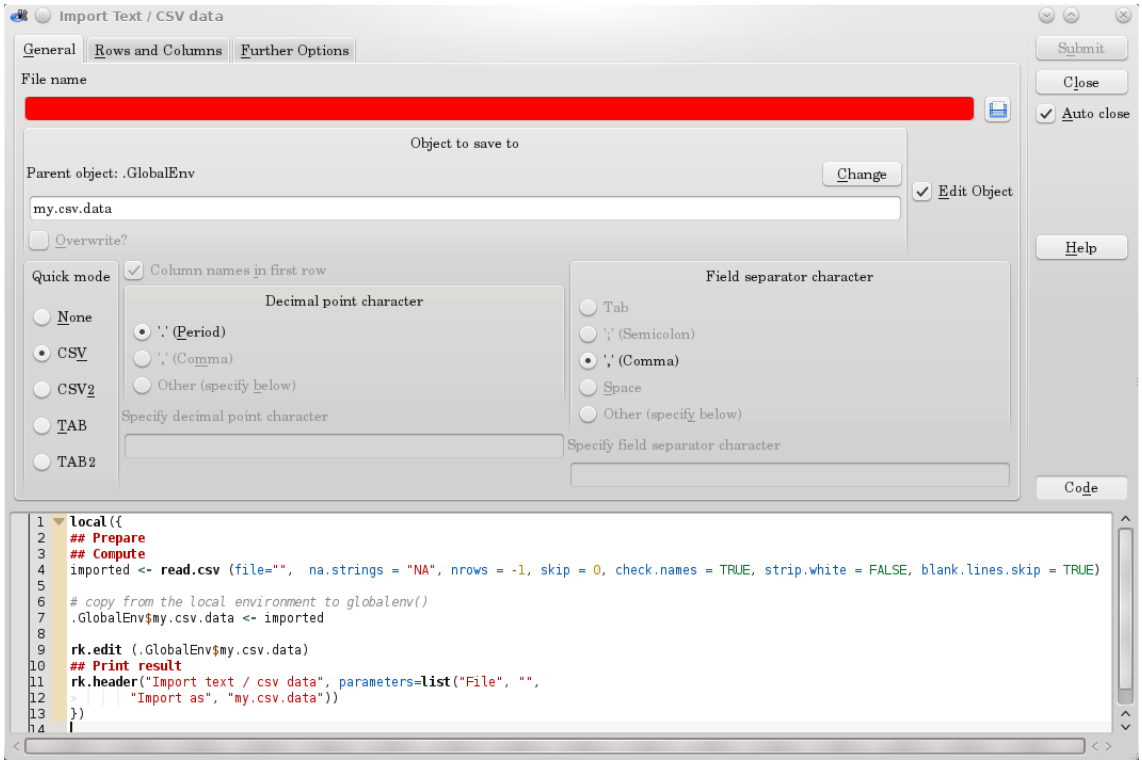

Figure 6: General data import dialog. Useful defaults for a variety of formats can be set using the "Quick Mode" selector on the left. Further customizations can be done from the "Rows and Columns" and "Further Options" tabs. The code in the bottom area can be copied and used for other purposes.

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.

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,

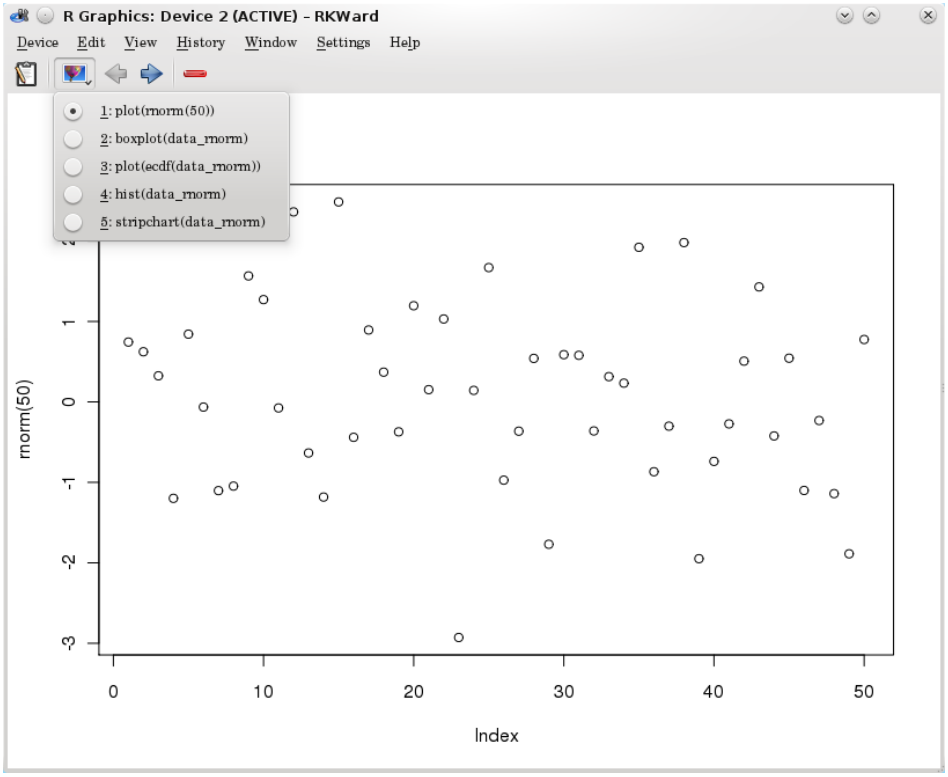

Figure 7: On-screen graphics device window in **RKWard**. The plot history is available as a drop-down list, allowing to jump directly to a previous plot. In this example, five different plots were performed on the same data set of a random sample (`rnorm()`). The plot can be exported via "Device→Export"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

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[13]. 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[14]. 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. 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[15] (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** (**?**).

The formatting of the 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 (~/.rkward, by default). 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** (**??**).

Images contained in the output are stored as portable network graphics (PNG)[16] by default, but JPEG[17] and scalable vector graphics (SVG)[18] 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 are not properly rendered by older supported versions of the **KDE** libraries.

[13] The preview is updated asynchronously to keep the GUI responsive; see Section **??**.
[14] Note: The font size of the output can be adjusted from the menu.
[15] In case not all parameters could be reused, since, may be, some of the objects in question are no longer available, the user will be notified.
[16] `http://www.libpng.org/pub/png/`
[17] `http://www.jpeg.org/jpeg/index.html`
[18] `http://www.w3.org/Graphics/SVG/`

RKWard output initialized on Fri Nov 12 07:54:43 2010

Descriptive statistics

Parameters

- Trim of mean: 0

Sun Nov 21 22:59:02 2010

Object	mean	median	min	max	standard deviation	length of sample	number of NAs
density	0.7191590909	0.5265	0.011	2.003	0.5955725771	176	0
conc	3.106689452	1.171875	0.04882812	12.5	4.059865328	176	0
weight	121.8183391	103	35	373	71.0719596	578	0
Time	10.71799308	10	0	21	6.758399725	578	0

Run again

Figure 8: Sample contents of the output window. `DNase` data of the **datasets** package was used. 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.

3.8. Package management

The number of R packages available from the comprehensive R archive network (CRAN), Omegahat[19] and Bioconductor (**?**) has grown exponentially since R 1.3 (2001) to R 2.7 (2008) (**???**). **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 (**??**).

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 this 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 (**?**).

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

[19] http://www.omegahat.org/

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 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[20]. 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. An 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.

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").

[20] For technical background of **RKWard** GUI help pages please refer to Section 5.1.1.

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") assists in reading the data by a common point & click interface (Figure 9A). 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).

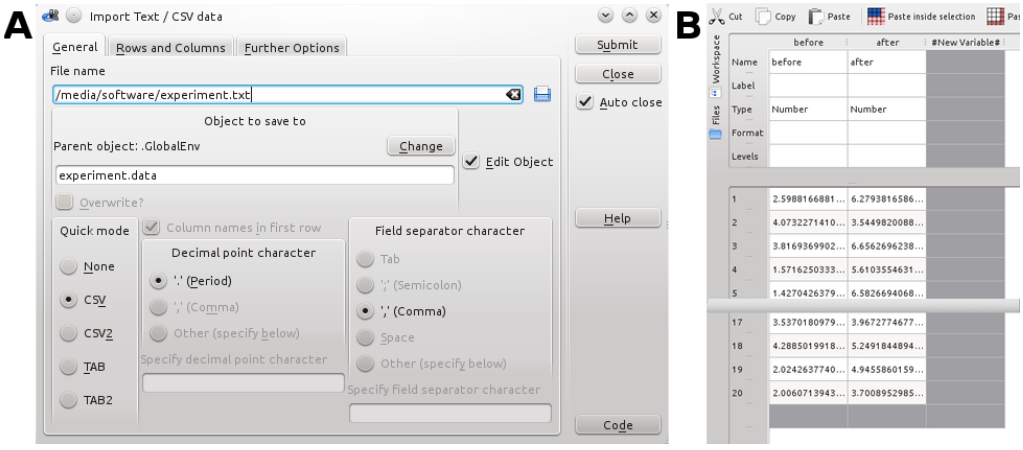

Figure 9: A) CSV import dialog. Useful defaults for a variety of common text separated value formats can be set using the "Quick Mode" selector on the left. Beyond that, many options can be customized. B) Data editor. The imported CSV data from "experiment.txt" are presented (data visually trimmed).

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 10A). Pressing the "Submit" button conducts the test, and opens the output document tab showing the results (Figure 10B).

4.3. Creating a plot

To visualize the data, "Boxplot" is chosen from the "Plots" menu and the two variables, corresponding to the t-test above, are selected. The dialog allows to define custom variable

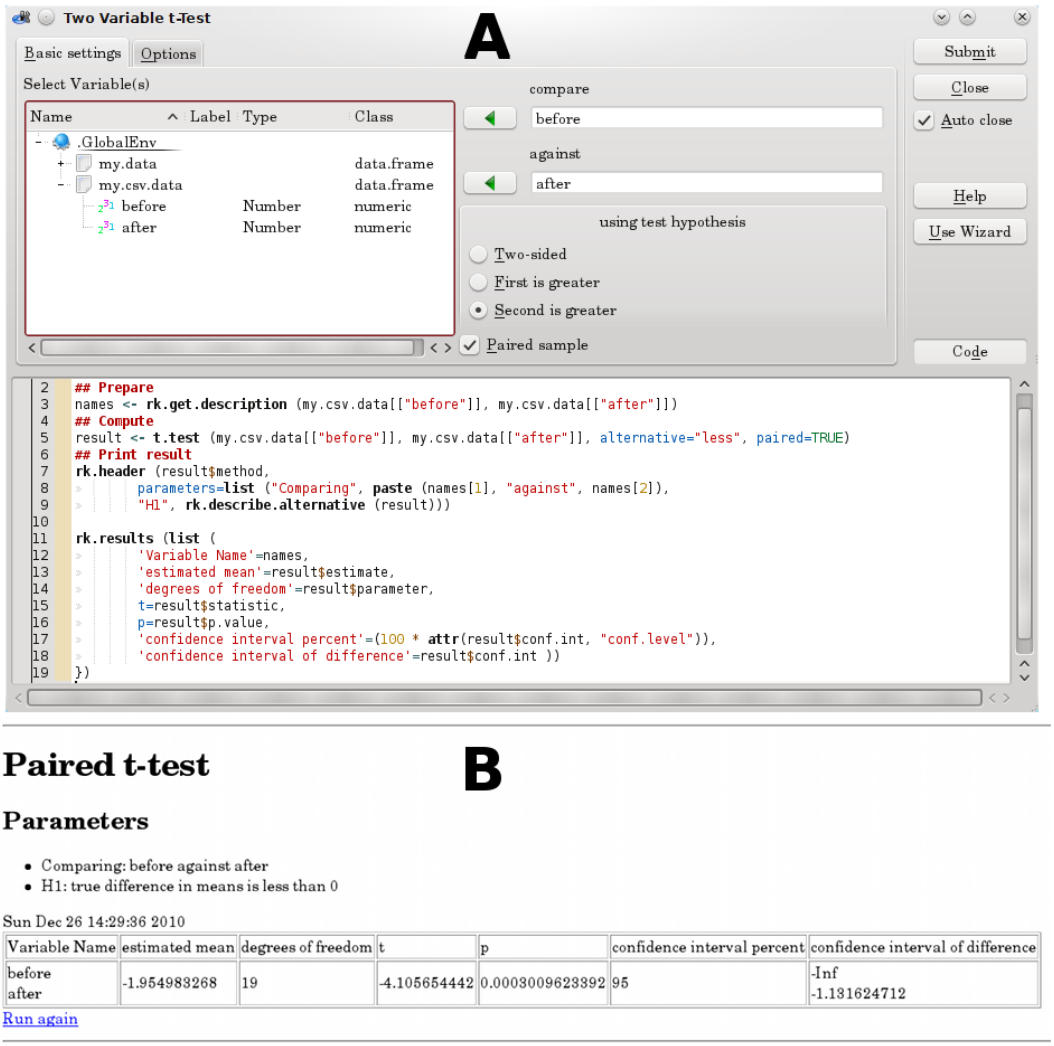

Figure 10: A) Student's t-test dialog for two variables. The bottom area shows the R code corresponding to the settings. B) Test results in tabular HTML format. Besides the result, information such as the date of analysis and the relevant test parameters are also reported.

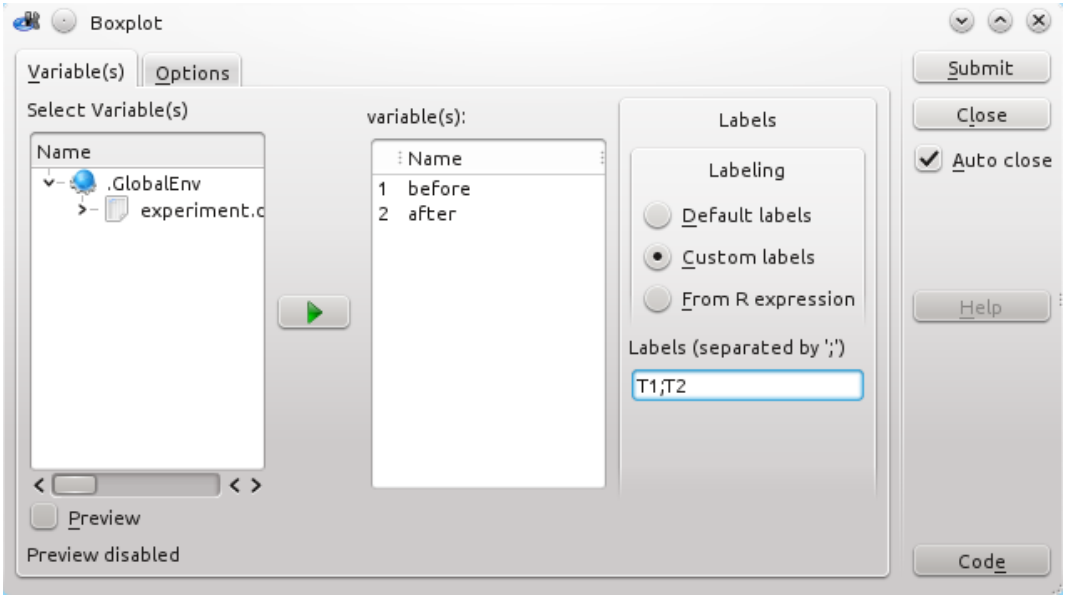

Figure 11: Boxplot dialog. The first tab "Variables" is used to select the variables for analysis. It is possible to combine any data present in `.GlobalEnv`. The second tab "Options" allows further adjustments (e. g., the addition of mean and standard deviation) to the plot (not shown).

labels (Figure 11). 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 12).

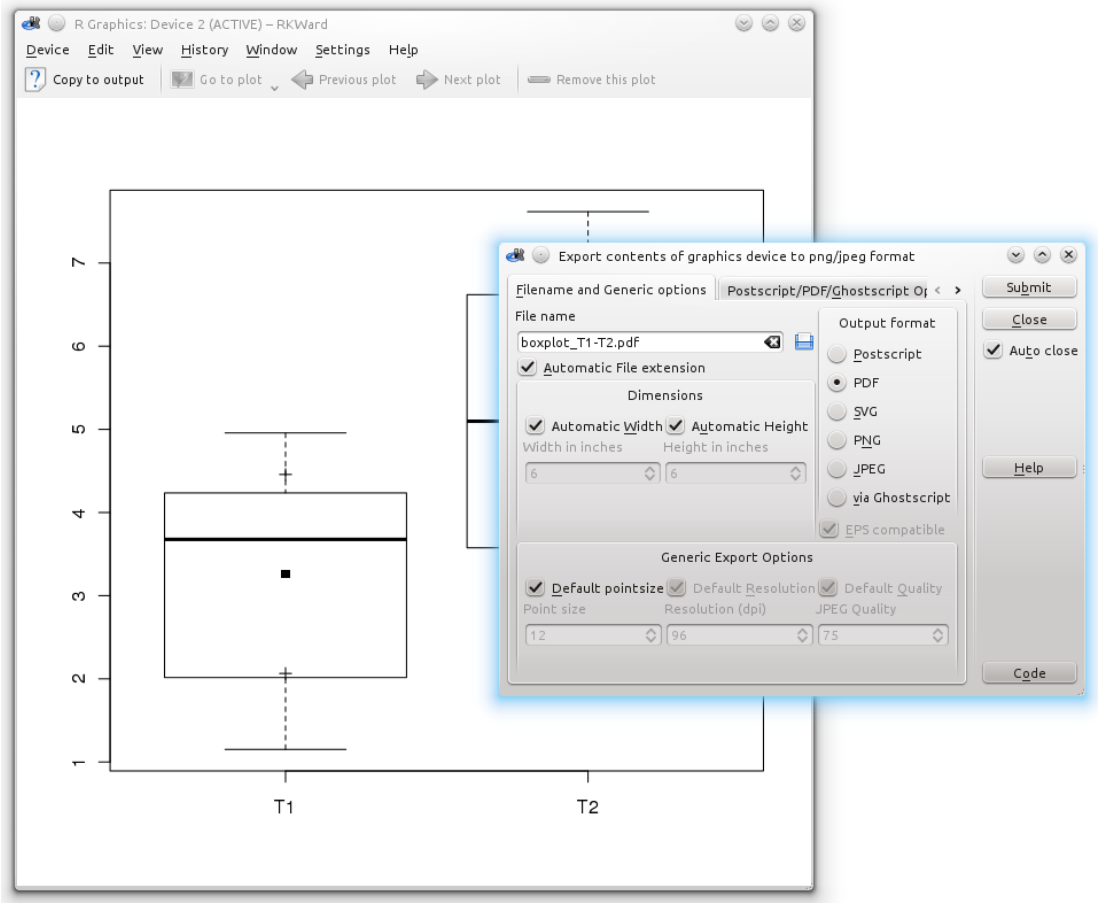

Figure 12: Plotted data and plot export dialog. The export dialog ("Device→Export") provides numerous options like resolution and size for different vector formats (e. g., SVG, PDF) and pixel formats (e. g., PNG, JPEG). (Note: For the shown figure, the optional mean (■) and standard deviation (+) parameters were selected in the boxplot plugin.)

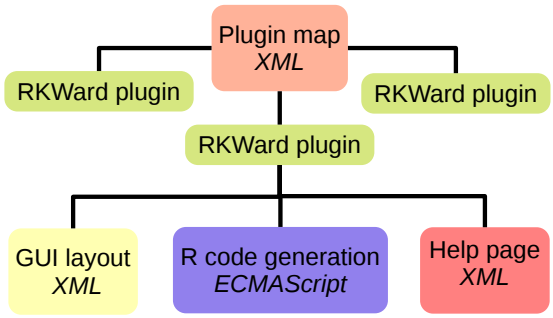

Figure 13: Plugin structure of **RKWard**. One or more plugins are declared in a "plugin map". Each plugin is defined by two XML files, and one ECMAScript file.

5. Fundamentals of the plugin framework

In this section we will give a brief overview of the design aspects of **RKWard**'s plugin framework, since this is central to the extensibility of **RKWard**. Further details can be found in the annex accompining this paper.

5.1. 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[21]. 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.

Defining a plugin

Plugins consist of four parts as visualized in Figure 13 (see Section 6 for an example; for a complete manual, see **?**):

- An XML file (Section 6.1), 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 6.2). 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

[21] 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.

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 6.3) is used to translate GUI settings into R code[22]. 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 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**.

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).

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

[22] 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.

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 (**?**) 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.

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)[23]. 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 (see Section 3.8) 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.

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

6. Extending RKWard – an example of creating a plugin

As discussed in Section 5.1, plugins in **RKWard** are defined by four separate files (Figure 13). 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 t-test. For brevity, the help-file is omitted.

6.1. 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 t-test (see Figure 14). 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>
```

6.2. 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 t-test plugin (`<!DOCTYPE rkplugin>`) to be organized in two tabs (Figure 10A). 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>
```

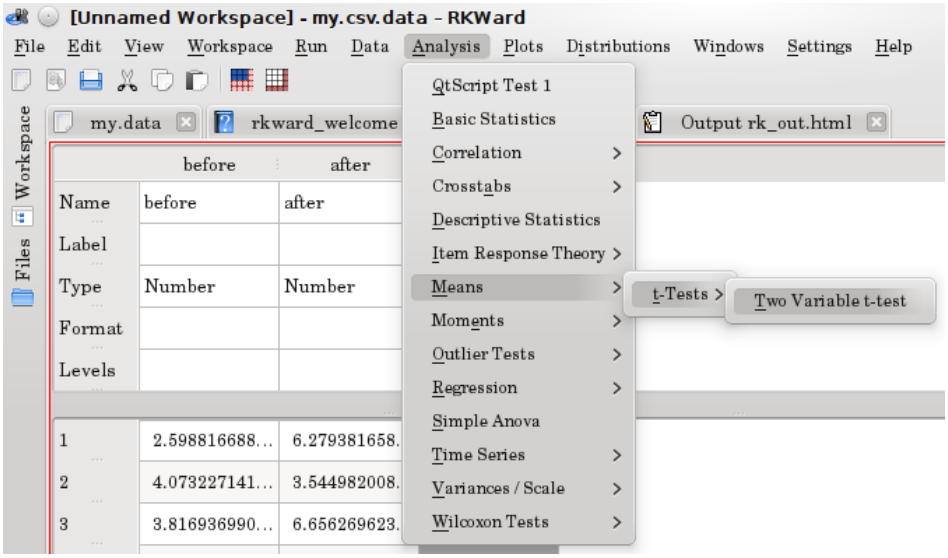

Figure 14: Generated menu structure as defined by the plugin map.

```
<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>

<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>
```

6.3. Generating R code from GUI settings

A simple ECMAScript script is used to generate R code from GUI settings (using `echo()` commands)[24]. 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');
```

[24] See Figure 10A) for code generated in this example.

```
  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');
}
```

7. 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.
- Uses standard R formats (see **?**) 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.

8. 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.

Affiliation:

Stefan Rödiger
Lausitz University of Applied Sciences
Department of Bio-, Chemistry and Process Engineering
AND
Center for Cardiovascular Research (CCR)
Charité, Germany
E-mail: stefan_roediger@gmx.de
E-mail: rkward-devel@lists.sourceforge.net

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd