RKWard plugin development with the `rkwarddev` package

m.eik michalke

October 20, 2011

Writing plugins for `RKWard` means writing at least two XML files (a GUI description and a plugin map), one JavaScript file (to create the `R` code), maybe a help file (again in XML), and for plugins who should be distributed, a DESCRIPTION file. Furtermore, all of these files need to be in a certain directory structure.

The `rkwarddev` package aims to simplify this, by enabling you to fulfill all the listed tasks in just one `R` script.

1 About the package

You might ask why you should write R scripts to generate plugins, if you could just directly write the XML and JavaScript files. First of all, you don't have to use this package at all, it's totally fine to code your plugins however you like. The main reason why I wrote this package is that I like to really concentrate on what I'm doing, so this is my attempt to avoid the need to switch between three different languages all the time. I wanted to be able to constantly "think in `R`" while working on a plugin. As a side effect, a lot of useful automation was implemented, and using this package will definitely save you quite some amount of typing.

2 Before we start

It is important to undertsand that while `rkwarddev` can help you to make designing new plugins much easier, you still need to know how the generated XML and JavaScript files work and interact. That is, if you didn't yet read the *Introduction to Writing Plugins for RKWard*,[1] please do so before you start working with this package. Once you're sure you understand how plugins in `RKWard` actually work, just come back here.

[1] `http://rkward.sourceforge.net/documents/devel/plugins/index.html`

3 Ingredients

If you look at the contents of the package, you might feel a little lost because of the number of functions. So let's first see that there's actually some order in the chaos.

Most functions start with the prefix `rk.` to indicate that they somehow belong to `RKWard` (we'll get to the exceptions later). After that, many names have another abbreviation by which they can roughly be classified into their specific "area" of plugin development:

- `rk.XML.*()`: XML code for GUI description (and plugin maps)
- `rk.JS.*()`: JavaScript code
- `rk.rkh.*()`: XML code for help pages

In short, you should find a `rk.XML.*()` equivalent to every XML tag explained in the *Introduction to Writing Plugins for RKWard*,[2] e. g. `rk.XML.dropdown()` to generate a `<dropdown>` menu node. There are a few functions for JavaScript generation which fall out of this scheme. That is because firstly they should be intuitively to use just like their JavaScript equivalent (like `echo()`), and secondly they are likely to be used very often in a script, so short names seemed to be a blessing here (like `id()` or `qp()`).

Adding to that, there are some special functions, which will all be explained later, but here's the list, roughly ordered by the development stage they're used for:

- `rk.paste.JS()`: Paste JavaScript code from `R` objects
- `rk.XML.plugin()`: Combine XML objects into one plugin GUI object
- `rk.JS.scan()`: Scan a GUI XML file (or `R` object) and generate JavaScript code (define all relevant variables)
- `rk.JS.saveobj()`: Scan a GUI XML file (or `R` object) and generate JavaScript code (save result objects)
- `rk.JS.doc()`: Combine JavaScript parts into one plugin JavaScript file object
- `rk.rkh.scan()`: Scan a GUI XML file (or `R` object) and generate a help page skeleton
- `rk.rkh.doc()`: Combine XML objects into one help page object
- `rk.testsuite.doc()`: Paste a testsuite skeleton
- `rk.XML.pluginmap()`: Combine XML objects into one plugin map object
- `rk.plugin.skeleton()`: Generate actual plugin files from the plugin GUI, JavaScript, help page, testsuite and plugin map objects (i. e., put all of the above together)
- `rk.build.plugin()`: Compress the generated files into an installable `R` package for distribution

[2] `http://rkward.sourceforge.net/documents/devel/plugins/index.html`