Generating S and C code with the RGtkBindingGenerator package

Duncan Temple Lang
Bell Labs

September 30, 2002

This `RGtkBindingGenerator` package generates S language bindings for Gtk-based C libraries. By "Gtk-based", we mean code that has Gtk classes and obeys the basic conventions of the Gtk library. This includes libraries such as Gtk itself, the Gnome libraries, gtkhtml, gtk+-extra, zvt, the Gtk Mozilla Embedded library, and so on.

The contents of these Gtk-based libraries are often described via .defs files which give a symbolic definition of the classes, routines, enumerations they contain. We read these definitions into R and generate both S and C interface functions and routines to provide an S interface to the library. All that remains for the author of the interface to do is

- write a header file that includes the basic header files for the C library to which we are interfaceing;
- write a makefile and configuration file to find the necessary header files and libraries and build the S library.
- if necessary, add any utility functions that are not in the Gtk-based library.

The basics of using this package are quite simple. One loads the package using the familiar *library()* function

1a ⟨ 1a⟩≡
```
library(GtkAutoBindingGen)
```

That is all that is needed to call our highest level function that reads the .defs files and generates the S and C code. This function is *generateCodeFiles()* and it takes an arbitrary collection of .defs files via its ... argument. Additionally, we can tell it where to place the S and C code files that it creates. We do this via the *dir* argument and we can specify a single directory for both sets of files (S and C code) or a pair of directories to be used for the S and C code separately. If we want the S and C code to be located in different directories, we pass a length 2 character vector with the directory for the C code first. We can also give the elements names: "C" and "S". (Use no other names for the the first two elements!)

To illustrate the idea, we will use the .defs file for the Zvt library which we include in the `examples/` directory of the package. The Zvt library provides an interactive terminal widget. We will tell the binding generator code to put the resulting files in two different directories: `/tmp/RZvt/src/` for the C code and `/tmp/RZvt/R/` for the S code. This follows the layout of an R package and so we will be able to easily build the package without having to move files around manually. So we start by creating the top-level directory of the package which we will call `RZvt`. This is where the `DESCRIPTION` and configuration files live in the package structure.

1b ⟨ 1a⟩+≡
```
dir.create("/tmp/RZvt")
```

Note that putting the files into the R package layout, should we need to regenerate the bindings (e.g. because we update the .defs file), the newly generated S and C code will overwrite the previously generated bindings and we can simply rebuild the package.
Now we are ready to generate the binding code. We call *generateCodeFiles()* and give it the location of the `zvt.defs` file where it will find the definitions. We also tell it where to place the C and S code using the *dir* argument. And we also tell it the name of the package for which this code is being generated, namely `RZvt`. This is used in the *.Call()* code that allows us S to invoke the C routines in the package. For S-Plus, this is not necessary but should be included .

⟨ 1a⟩+≡
```
generateCodeFiles(system.file("examples", "zvt.defs", package="RGtkBindingGenerator"),
                  dir=c("/tmp/RZvt/src/","/tmp/RZvt/R"), package="RZvt")
```

This will generate 11 files: 6 C files in `/tmp/RZvt/src/`, and 5 S files in `/tmp/RZvt/R/`. The S files are:

zvtAccessors.R	functions to read the fields or slots of a Gtk object.
zvtEnumDefs.R	S vectors mirroring the C level defintions of any enumerations or flags in the .defs file. These are nar
zvtEnumFuncs.R	S functions for validating or computing the value of any enumerations or flags defined in the ..defs fi
zvtFuncs.R	defintions for the S functions corresponding to the routines defined in the `zvt.defs` file.
zvtzConstructors.R	defintions for the S constructor functions for the different classes in the library.

For each of these, there are corresponding C source and declaration/header files.

zvtFuncs.c	source code for the interface routines that are called from S functions than in turn call the C routines in t
zvtFuncs.h	declarations of the routines in `zvtFuncs.c`. These declarations aren't necessary, but can be used if we wa
zvtEnum.c	C routines that check an S value as being valid for a given enumeration or flag definition. These check wh
zvtEnum.h	the header file providing the declarations for these routines. Again, this is useful for creating registration
zvtAccessors.c	C code that provides the wrapper routines for reading the values of fields in a Gtk objects in this library.
load.c	C code for forcing the explicit instantion of each class within this library. This is useful when we want to

Now that we have this code, we need to organize it into an R package. This involves compiling and linking the C code as a shared library/DLL.

First, we create a local C header file for the package, `RZvt.h`. This is included by the automatically generated C files and it is expected to provide the definitions for the Zvt library and any other information that might be necessary to compile code that uses that library. Specifically, it must include the `zvtterm.h` header file.

⟨ 1a⟩