HowTo use the widgetInvoke package

Jeff Gentry

April 30, 2008

1 Overview

The *widgetInvoke* package, available as part of Bioconductor, is designed to allow for the creation of user interfaces for arbitrary functions. The design involves the use of Gtk for the actual interface and XML for data exchange, both of which were chosen for their cross platform abilities. Package authors and maintainers can create and customize metadata files for the functions of their packages using tools provided by *widgetInvoke* (or on their own), distribute these files with the package and then users of the package can use *widgetInvoke* to provide a GUI interface to this function. With all of this, a simple and standare mechanism can be used to quickly allow for GUI interaction with the functions of any given package.

2 Getting Started

The *widgetInvoke* package requires the use of the *RGtk* package from `www.omegahat.org` to interface with the Gtk libraries. While not explicitly required, the *RGtkHTML* and *RGtkDevices* are recommended as it will provide a bit richer environment (although at this time, users will not encounter any functionality loss if they are not installed, just a lower quality interface in a couple of spots).

To load the *widgetInvoke* package, use the `library` function:

```
> library(widgetInvoke)
```

Also note that for the examples in this vignette, we shall be using a copy of the `apropos` function, named `testWIfun`.

3 The `widgetInvoke` function

The `widgetInvoke` function is used by the end user of a package to present the GUI for their desired function. To do this, they first must load not only the *widgetInvoke* package, but make sure that the function they wish to actually use is also loaded. At this point, the user only needs to call `widgetInvoke` with the name of the function they wish to use.
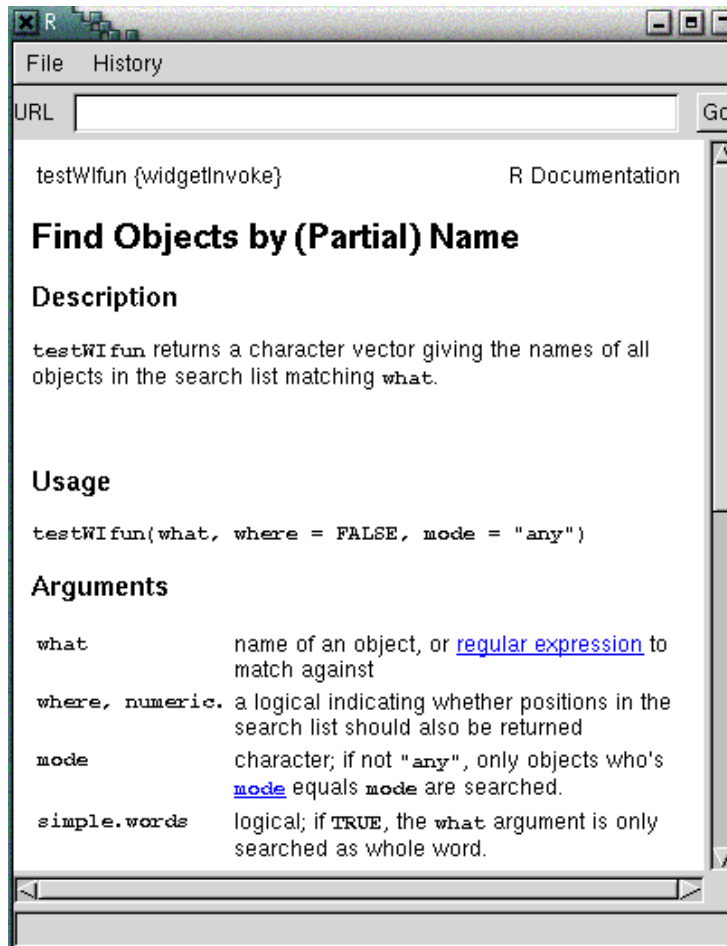
```
> if (interactive()) {
+      widgetInvoke("testWIfun")
+ }
```
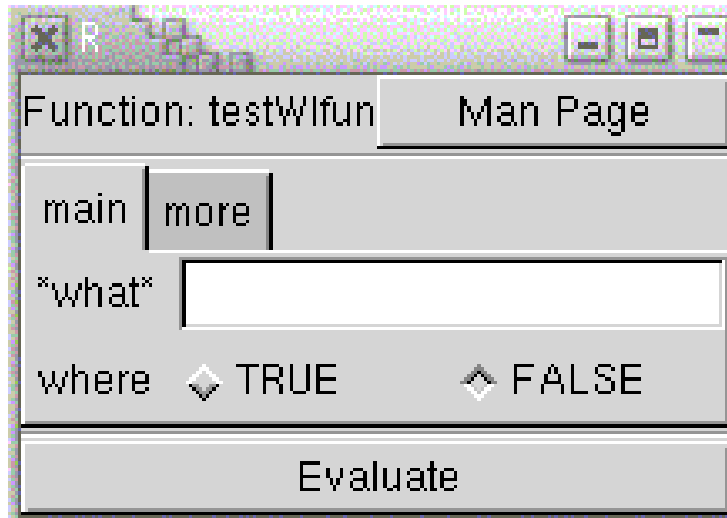

Please note that this and most other code chunks shall have the 'interactive'
test. This is solely for the handling of this vignette by automatic sessions without
displays, and users following along do not need to include this test (although it
will not hurt to do so).

Here we are presented with the interface window of a very simple function,
testWIfun (which of course is an exact replica of apropos). This function has
three arguments (what, where and mode) which in this case are spread across
two separate notebook panes ('main' and 'more'). The assignment of arguments
to a particular notebook pane is done by the creator of the metadata (typically
the package author or maintainer) and is explained in the section 'Creating the
function metadata'.

The function name is presented at the top, with a button which will display
the man page for this function (using *RGtkHTML* if available, otherwise a less
rich text widget is used). Clicking on this button will provide a window like
this:

```
testWIfun {widgetInvoke}                          R Documentation
```

Find Objects by (Partial) Name

Description

testWIfun returns a character vector giving the names of all objects in the search list matching what.

Usage

testWIfun(what, where = FALSE, mode = "any")

Arguments

what	name of an object, or regular expression to match against
where, numeric.	a logical indicating whether positions in the search list should also be returned
mode	character; if not "any", only objects who's mode equals mode are searched.
simple.words	logical; if TRUE, the what argument is only searched as whole word.

The what argument has its name surrounded by '*' characters (*what*). This implies that the what function is required to have a value (default values are okay, but this function does not provide one) before evaluation. Other arguments are okay to leave empty if that is your desire. Here we will use "testWIfun" as the what argument, select TRUE for the where argument (instead of using the default value of FALSE), and in the second pane we shall stay with the default value of "any" for the mode argument. When these are filled in, we can hit the Evaluate button:
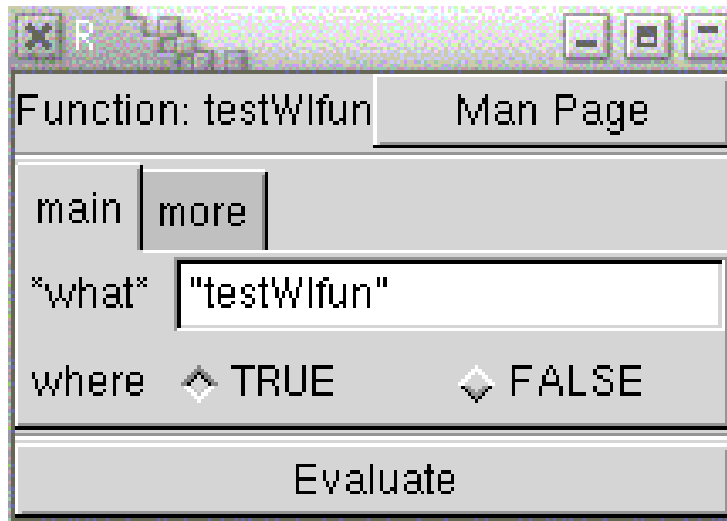
This leaves us with the output:

```
        2
"testWIfun"
```

As you can see, the function was evaluated in the R session as if the user called `testWIfun` themselves.

One note on the entry of values into the text entry fields. Anything entered in there is handled in almost the same manner as entering a value into R - if the value is quoted (e.g. `"any"`), it is treated as a character string. If the value is not quoted, it is first checked to see if it represents an object in the search path and if it is not then it is assumed to be of numeric type. The one exception to this is a comma separated list of values, which are handled as a vector (where each element is handled by the rules above). For complex values, it is recommended to assign them first to a variable before calling `widgetInvoke` and then using that variable in the entry field. Arguments which have limited possiblities for values use radio buttons or drop down lists, and do not have this problem.

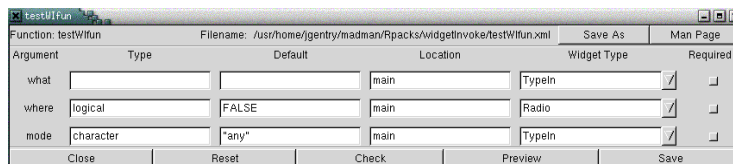## 4   Creating the function metadata

The `widgetInvoke` function will not work unless the proper structure has already been put in place to support a given function in a package. This must be done ahead of time, typically by the package author or maintainer. To do this, a metadata file must be created for every function that is intended to work with `widgetInvoke` and stored in the `inst/wFun` directory of the package sources. This file is in XML format, and that format is discussed in the section 'The widgetInvoke XML'.

There is an authoring tool in the *widgetInvoke* package that provides an easy interface to generating these metadata files, the `createWF` function. This will

first attempt to figure out as much information as possible about the desired function, either from the function itself or if a metadata file already exists in the current directory it will use that for default values, and then display an interface to allow the operator to modify the settings for the display.

To do this, simply call the `createWF` function with the name of the function that you would like to create the metadata for:

```
> if (interactive()) {
+     createWF("testWIfun")
+ }
```

Argument	Type	Default	Location	Widget Type	Required
what			main	TypeIn	
where	logical	FALSE	main	Radio	
mode	character	"any"	main	TypeIn	

Function: testWIfun — Filename: /usr/home/jgentry/madman/Rpacks/widgetInvoke/testWIfun.xml — Save As — Man Page. Buttons: Close, Reset, Check, Preview, Save.

The window itself is structured in a tabular format. At the top is the name of the function, the current name of the file to save to (using the `Save` button), a `Save As` button to change that filename and a button to provide the man page (which works identically to the one in the `widgetInvoke` function). Below the table are five buttons: `Close`, `Reset`, `Check`, `Preview` and `Save`. The table in the middle provides a row for every argument with columns marked `Type`, `Default`, `Location`, `WidgetType` and `Required`.

The `Close` button will simply close the window without performing any further work. The `Reset` button will reset the fields in the argument table to the values that existed when this instance of `createWF` was started. The `Check` button will attempt to check the validity of the current values in the argument table, and alert the user to any problems that might be detected, if there are no problems a dialog will appear detailing that there are no problems. The `Preview` button will display a sample of the window that a user of the `widgetInvoke` function would see - in this instance the `Evaluate` button of this subwindow will simply close the window and not actually perform any call to R. Lastly the `Save` button will save the values in the argument table to the XML metadata file.

In the argument table, each row corresponds to an argument, and each column corresponds to part of the metadata required for the use of this function with `widgetInvoke`. The first column, `Type`, details what type the value for this argument should be, e.g. `logical`, `character`, `numeric`, etc. For arguments that do not have a specific type requirement, the `ANY` value should be used. When `createWF` attempts to determine an appropriate default, if there is no default value for a particular argument it will automatically assign `ANY`, so operators of `createWF` should double check that this is actually true. The `Type` can also be a character vector, which corresponds to a structure such as `xloc=c("equispaced", "physical")`. In a situation like this, the first value of the vector is assumed to be the default, which is another situation where the operator should double check this value.

The `Default` field will specify the default value for the argument. If one is specified in the formal arguments for the target function, it will appear here, but otherwise there will not. Operators of `createWF` can assign a new default or remove the default altogether. The type of the default value should match the type listed in `Type`, and this will be verified by the `Check` button.

The `Location` field specifies which notebook pane this argument will appear on. These are sorted by name, and in a fresh run of `createWF` will all be set to `main`. To create a new pane, simply create a new name for the `Location` field of a particular row. There is no limit to the number of arguments that can be in a particular pane.

The `WidgetType` field allows the operator to specify what sort of display widget that `widgetInvoke` uses for this argument. There are currently three options: `Radio` which uses radio buttons, `DropDown` which uses a drop down list, and `TypeIn` which provides a text entry widget. The first two (`Radio` and `DropDown`) are limited to arguments with a set of possible values - arguments with the `Type` set to being `logical` or a vector of possible values. All other arguments currently must use the `TypeIn` widget. These rules are enforced when the `Check` button is used.

The `Required` button, if checked, indicates that this particular argument must be filled in by the `widgetInvoke` user of this function before evaluation can take place (or they can use the default value, if one is provided). In a fresh instance of `createWF` for a function, these are all unchecked. Operators should make sure that any argument that is crucial to the proper flow for this function are marked as being required.

Once all of the fields are written to the operator's satisfaction, the `Preview` button can be used to make sure that the layout is visibly appealing as well. The `Check` button should be used to make sure that there are no problems with the defined structure, and if all is well the `Save` button can be used to save this particular metadata to a file named `function.xml` (where `function` is the name of the function) and stored in the current working directory of the operator's R session. For distribution, this should be put in the `inst/wFun` directory of the package. At this point, users can interact with this function with `widgetInvoke`.

5   The *widgetInvoke* XML

The XML format used by the *widgetInvoke* package is simple and straightforward. The `wFun` tag defines the primary XML block that is specifying the `wFun` object for a particular function. Every function has its own file, and thus a single `wFun` block.

Within the `wFun` block are two main children nodes, `funName` and `funArgList`. The former simply provides for the name of the function, while the latter contains a series of `funArg` blocks - one for each argument in this function. The `funArg` blocks contain the majority of the information stored in the `wFun`, each one providing six fields:

argName: Name of this argument.

```
argDefault:   The default value, if any, of this argument.
argType:      The type of this argument (e.g. logical, numeric,
              character).  "ANY" is used for typeless arguments.
argLocation:  The name of the notebook pane this argument will be
              displayed in.
argRequired:  A logical value specifying whether or not this argument
              is required or not.
```

Each of these are then stored in the inst/wFun directory of the package source, using the name function.xml (where 'function' is the name of the function). When installed, these files are available in the wFun directory of the installed package.

An example of the XML is provided for the testWIfun function.

```
<?xml version="1.0"?>
 <wFun xmlns:bt="http://www.bioconductor.org/WINVOKE">
        <funName>testWIfun</funName>
        <funArgList>
                <funArg>
                        <argName>what</argName>
                        <argDefault></argDefault>
                        <argType>ANY</argType>
                        <argLocation>main</argLocation>
                        <argWidgetType>TypeIn</argWidgetType>
                        <argRequired>TRUE</argRequired>
                </funArg>
                <funArg>
                        <argName>where</argName>
                        <argDefault>FALSE</argDefault>
                        <argType>logical</argType>
                        <argLocation>main</argLocation>
                        <argWidgetType>Radio</argWidgetType>
                        <argRequired>FALSE</argRequired>
                </funArg>
                <funArg>
                        <argName>mode</argName>
                        <argDefault>"any"</argDefault>
                        <argType>character</argType>
                        <argLocation>more</argLocation>
                        <argWidgetType>TypeIn</argWidgetType>
                        <argRequired>FALSE</argRequired>
                </funArg>
        </funArgList>
</wFun>
```