# Package: svIDE (via r-universe)

July 2, 2024

**Type** Package

**Version** 0.9-54

**Date** 2018-06-28

**Title** Functions to Ease Interactions Between R and IDE or Code Editors

**Author** Philippe Grosjean [aut, cre]

**Maintainer** Philippe Grosjean <phgrosjean@sciviews.org>

**Depends** R (>= 2.6.0)

**Imports** utils, tcltk, svMisc, XML

**Description** Function for the GUI API to interact with external IDE/code editors.

**License** GPL-2

**URL** http://www.sciviews.org/SciViews-R

**BugReports** https://r-forge.r-project.org/tracker/?group_id=194

**NeedsCompilation** no

**Date/Publication** 2018-06-28 13:45:20 UTC

**Repository** https://phgrosjean.r-universe.dev

**RemoteUrl** https://github.com/cran/svIDE

**RemoteRef** HEAD

**RemoteSha** d01e59a1ede17f26bac7676422466cacad16ef6d

## Contents

---

`svIDE-package` *Functions to Ease Interactions Between R and IDE or Code Editors*

---

## Description

Function for the GUI API to interact with external IDE/code editors.

## Details

| | |
|---|---|
| Package: | svIDE |
| Type: | Package |
| Version: | 0.9-54 |
| Date: | 2018-06-28 |
| License: | GPL 2 or above, at your convenience |

## Author(s)

Philippe Grosjean

Maintainer: Ph. Grosjean <phgrosjean@sciviews.org>

---

`createSyntaxFile` *Create a syntax definition or a calltip file for R language*

---

## Description

A .svl syntax file describes the syntax of the language for SciViews GUIs. It is used mainly for syntax coloring of text in editors. The calltip file (by default, Rcalltips.txt) is a database with formal calls of R functions, to be used by code editors to display function calltips.

## Usage

```
createSyntaxFile(svlfile = "R.svl", pos = 2:length(search()))
createCallTipFile(file = "Rcalltips.txt", pos = 2:length(search()),
    field.sep = "=", only.args = FALSE, return.location = FALSE)
```

## Arguments

| | |
|---|---|
| `svlfile` | the name or location of the .svl file you want to create. |
| `file` | the name or location of the calltip file you want to create. |
| `pos` | a vector of integers indicating which positions in the search path should be recorded in the file. |
| `field.sep` | the field separator to use between the function name and its calltip in the file. |
| `only.args` | do we record the full calltip (`myfun(arg1, arg2 = TRUE, ...)`), or only the function arguments (`arg1, arg2, ...`). |
| `return.location` | when `TRUE`, the package where this function is located in returned between square brackets at the end of the line. |

## Value

These functions return nothing. They are invoked for their side effects of creating a file.

## Note

SciViews-R uses a file named 'R.svl' and located in <SciViewsDir>/bin/languages. This function generates such a file. Do resist to the temptation to generate a very long list of keywords by loading many packages. SciViews cannot handle a list longer that 32k, that is roughly, 2000 - 2500 keywords.

`createCallTipFile()` sometimes issues warnings because it cannot get arguments from some keywords. You can ignore these warnings.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

`getFunctions`, `getKeywords`

## Examples

```
## Not run:
## Create a syntax highlighting file for all currently loaded R packages
createSyntaxFile("Rtemp.svl")
## Show and delete it
file.show("Rtemp.svl", delete.file = TRUE)

## Create a calltips file for all currently loaded R packages
createCallTipFile("Rtemp.ctip", return.location = TRUE)
## Show and delete it
file.show("Rtemp.ctip", delete.file = TRUE)

## You can also customize the calltip file and select the package
## Here we include only functions from base package (the last item
```

```
## in the search path)
createCallTipFile("Rtemp2.ctip", pos = length(search()),
    field.sep = ";", only.args = TRUE)
## Show and delete it
file.show("Rtemp2.ctip", delete.file = TRUE)

## End(Not run)
```

---

`getFunctions` *Get all functions in a given environment*

---

## Description

Get a list of all visible functions in a given environment.

## Usage

```
getFunctions(pos)
```

## Arguments

| | |
|---|---|
| `pos` | the position in the search path. |

## Value

A list of character strings with functions names.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

`getKeywords`, `createSyntaxFile`

## Examples

```
getFunctions(1)  # Functions defined in .GlobalEnv
length(getFunctions(length(search())))  # Number of functions in package:base
```

`getKeywords` *get all keywords for syntax highlighting*

## Description

Get all visible keywords in one or several environment, excluding operators and reserved keywords.

## Usage

```
getKeywords(pos = 2:length(search()))
```

## Arguments

| | |
|---|---|
| `pos` | a vector of integers with all positions in the search path where to look at. |

## Value

A vector of character strings with keywords.

## Note

This function is used by `createSyntaxFile()` to list all keyword2 items (thus excluding reserved keywords and operators).

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

`getFunctions`, `createSyntaxFile`

## Examples

```
getKeywords(1:2)
```

guiDDEInstall *install a DDE server (Windows only) for external IDE/code editor*

## Description

These functions install and manage a DDE server to return context-sensitive calltips or completion lists to external IDE/code editors under Windows.

## Usage

```
guiDDEInstall()
guiCallTip(code, file = NULL, onlyargs = FALSE, width = 60, location = FALSE)
guiComplete(code, file = NULL, sep = "|")
```

## Arguments

| | |
|---|---|
| code | a piece of R code (in a character string) to analyze. |
| file | a file where to return the result ("", or NULL for none). You can use "clipboard" to send the result to the clipboard under Windows only. |
| onlyargs | do we return the whole calltip or only the function arguments? |
| width | reformat the calltip to with (use 0 for not reformatting it). |
| location | if TRUE then the location (in which package the function resides) is appended to the calltip between square brackets. |
| sep | the separator to use between the item and its type in the list. |

## Value

These functions should be used to interact with an external program. Their result is returned invisibly for debugging purposes and is not intended to be use in R.

## Note

DDE is a communication protocol that exists only under Windows. Consequently, those functions cannot be used (yet) on other platforms.

On loading of the package, if the `option(use.DDE = TRUE)` is defined, the DDE server (`guiDDEInstall()`) is automatically installed when the package is loaded. Also if `options(IDE = "[path.to.exe]")` is defined, then that IDE is automatically started afterward.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

callTip, completion

## Examples

```
## Not run:
## DDE exchange protocol is available ONLY under Windows!

## Also, this cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

## Here is how you can test these features under Windows
options(use.DDE = TRUE)
library(svIDE) # This should automatically start the
## DDE server named 'TclEval SciViewsR', according to the option set

## Get some data in the user workspace
data(trees)
a <- 1
b <- "some text in the first instance of R"

########################################################
## To test these functions in an external program, we need now
## to start a second instance of R. In this second instance, enter:
library(tcltk)
.Tcl("package require dde")
.Tcl("dde services TclEval {}")
## You should get 'TclEval SciViewsR' in the list
## if the server in the first instance is running

## Now, request a calltip for the function 'ls'
## This is done in two steps:
## 1) Execute the command 'guiCallTip' with this code chunk as argument
.Tcl("dde execute TclEval SciViewsR {guiCallTip {res <- ls(}}")
## 2) Retrieve the calltip from the variable SciViewsR_CallTip
.Tcl("dde request TclEval SciViewsR SciViewsR_CallTip")

## Another way to trigger DDE commands (for programs that do not support
## DDE is to use 'execdde.exe' of the tcltk2 package (see ?tk2dde)

## It is also possible to copy the calltip to a file, or to the clipboard
## by specifying it after the command (also the two additional arguments
## have their default values changed)
.Tcl("dde execute TclEval SciViewsR {guiCallTip {library(} clipboard TRUE 40 TRUE}")
## Look at what is in the clipboard
cat(readClipboard(), "\n")

## The process is similar to get completion lists
.Tcl("dde execute TclEval SciViewsR {guiComplete {iris$}}")
.Tcl("dde request TclEval SciViewsR SciViewsR_Complete")

## Get the list of variables in the user workspace of the first R instance
## into the clipboard (use also the other arguments to get the type of objects)
.Tcl("dde execute TclEval SciViewsR {guiComplete {} clipboard TRUE { - }}")
## Look at what is in the clipboard
cat(readClipboard(), "\n")
```

```
## End(Not run)
```

---

kpfTranslate *Create a POT file or use a PO file to allow translating Komodo project (.kpf) or package (.kpz).*

---

## Description

Komodo Edit/IDE (code editor programs) use projects and packages for easy customization. However, these projects/packages are static elements that do not allow to use translation facilites easily. The `kpX2pot()` and `kpXTranslate()` functions provide a mechanism to use poEdit (a program designed to translate strings for software) to ease and automatise translation of such Komodo projets/packages (see details section for the procedure).

## Usage

```
kpf2pot(kpfFile, potFile)
kpz2pot(kpzFile, potFile)
kpfTranslate(kpfFile, langs, poFiles, kpf2Files)
kpzTranslate(kpzFile, langs, poFiles, kpz2Files)
```

## Arguments

| | |
|---|---|
| `kpfFile` | a Komodo project file with a .kpf extension to be translated. |
| `kpzFile` | a Komodo package file with a .kpz extension to be translated. |
| `potFile` | the name of a .pot file providing strings to be translated, if not provided, it is the same name as the .kpf/.kpz file, but with a .pot extension instead. |
| `langs` | the language(s) to translate to, for instance, 'fr' for French, 'de' for German, 'it' for Italian, 'en_GB' for Great Britain's English, etc. |
| `poFiles` | the path to the .po files containing the translations. If not provided, it is assumed to be the basename of the .kpf/.kpz file without extension, plus '-', plus langs and with a .po extension. For instance, the default translation file for `myproject.kpf` into French is named `myproject-fr.po`. |
| `kpf2Files` | the Komodo project files to create with the translations. If not provided, it is assumed to be the same as the initial project file, but with '-lang' appended to the base name. For instance, the French translation of `myproject.kpf` would be `myproject-fr.kpf` in the same directory, by default. |
| `kpz2Files` | same as kpf2Files, but for Komodo package files with a .kpz extension. |

## Details

Komodo Edit/IDE are code editor programs that can be used to edit R code efficiently with the SciViews-K plugin (see http://www.sciviews.org/SciViews-K). Komodo can be customized by using projects files (files with a .kpf extension), or tools collected together in the toolbox and that can be saved on disk in Komodo package files (with a .kpz extension).

Among the tools you can place in a Komodo project or package, there are macros (written in JavaScript or Python). Thanks to the SciViews-K plugin, you have access to R and R code inside these macros. This makes it a good candidate for writing GUI elements, including dialog boxes, on top of your favorite R code editor. You can also add 'snippets' in those projects/packages. Snippets are short pieces of code, including R code, you can save and retrieve easily. In the snippets, you can define replaceable parts, includings parts you replace after prompting the user for their values with a dialog box. SciViews uses these features extensively, for instance, for the 'R reference' toolbox (a kind of electronic reference card for R code).

Unfortunately, these tools do not benefit easily from translation features. So, it is hard to maintain the same project/package in different languages. The functions provided here ease the maintenance of such projects/packages translated in various languages. Here is how you can use them:

1) Save your Komodo project or package on disk (click on the project and use context menu to save it, or select all items you want to package in your toolbox and also use to context menu to create the package file).

2) Use the `kpX2pot()` function to create, or update a .pot file. This file lists all translatable strings found in the project/package. Translatable strings are: (a) names of tools or folders, (b) items in snippets that are flagged with %ask:R-desc:, %ask:R-tip:, %ask:URL-help:, %ask:RWiki-help:, %pref:URL-help:, %pref:RWiki-help, and %tr: (see Komodo help to learn how to use these tags in snippets), and (c) strings in JavaScript macros that are flagged with `_()`. For instance, creating a .pot file for `~/myproject.kpf` is as simple as calling `kpf2pot("~/myproject.kpf")`.

3) Use the poEdit program (search Google to find, download and install this free Open Source translation utility, if you don't have it yet) to translate the extracted strings. The first time, you create a .po file based on the .pot template you just created. For subsequent versions of your project/package, you reuse the old .po file and select menu entry 'Catalog -> Update from POT file...' in poEdit to update your translation file with new strings found in the recent .pot file. You are better to place the .po file in the same directory as your project/package and to give it the same name, but replacing .kpX by -<lang>.po, where <lang> is the language in which you do the translation. You can distribute .pot files to a staff of translators that would send you back the created/modified .po files for compilation. See the poEdit documentation for further help (note that multiline strings and singular/plural forms are not supported yet by `kpXTranslate()`).

4) Once you have your .po files ready, you can translate your Komodo project/package in these languages easily. For instance, a project file `~/myproject.kpf` can be translated in French, using the .po file `~/myproject-fr.po` and in Italian using a .po file `~/myproject-it.po`. To do so, you simply type `kpfTranslate("~/myproject.kpf")` in R. That produces a `~/myproject-fr.kpf` file that contains your Komodo project translated in French, and a `~/myproject-it.kpf` file with your Italian translation. Please, note that `kpzTranslate()` currently needs to access the external `zip` program for zipping the .kpz file. This program is usually accessible from within Linux or Mac OS X by default, but needs to be installed (and made accessible through the PATH) under Windows.

5) To open your translated project/package in Komodo, just drag and drop the new file in the central area of the Komodo window, and the project is open in the projects tabs at left, or the content of the package is added in you toolbox at right, depending on the type of file you use.

## Value

These functions return invisibly TRUE if the targetted files are created, or FALSE otherwise. Use `any(kpfTranslate("myfile.kpf"))` to check that ALL translations are done.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

---

`makeIconGallery` *Create galleries of icons for SciViews-K and Komodo*

---

## Description

The "pick icon" dialog box of Komodo uses icon galleries. SciViews-K adds several galleries for easier access to the icons, and it supplements about 1800 additional icons. The present function builds those galleries, based on a list of icons manually compiled (compatible with Komodo Edit/IDE version 9).

## Usage

```
makeIconGallery(flist)
```

## Arguments

| | |
|---|---|
| `flist` | the path to the ASCII text file containing the URIs of the different icons to collect together in the gallery. It is supposed to use the .txt extension, which is replaced by .html in the gallery file. |

## Value

TRUE (success) or FALSE (error) is returned invisibly.

## Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

## Source — *Source R code, capture its output and convert it in a different format*

### Description

This function is deprecated in favor of `sourceFormat()` and will disappear in the future version 1.0 of this package ! `Source()` is like `source()`, but it allows to rework the output (for instance to print it in HTML format).

### Usage

```
Source(...)
```

### Arguments

| | |
|---|---|
| `...` | Same arguments as for the `sourceFormat()` function. |

### Details

This function is usually called by functions that processes commands send by GUI clients.

### Value

The formatted output is returned invisibly.

### Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>), after code written by Tom Short

### See Also

`sourceFormat`, `source`

## sourceFormat — *Source R code, capture its output and convert it in a different format*

### Description

`sourceFormat()` is like `source()`, but it allows to rework the output into a different format (for instance to print it in HTML format).

### Usage

```
sourceFormat(file, out.form = getOption("R.output.format"), local = FALSE,
    echo = FALSE, print.eval = TRUE, verbose = getOption("verbose"),
    prompt.echo = getOption("prompt"), max.deparse.length = 150,
    chdir = FALSE, prompt = FALSE)
```

## Arguments

| | |
|---|---|
| file | a connection or a character string giving the name of the file or URL to read from. |
| out.form | a string indicating which output format to use (for instance, "html"). |
| local | if 'local' is 'FALSE', the statements scanned are evaluated in the user's workspace (the global environment), otherwise in the environment calling 'source'. |
| echo | logical; if 'TRUE', each expression is printed after parsing, before evaluation. |
| print.eval | logical; if 'TRUE', the result of 'eval(i)' is printed for each expression 'i'; defaults to 'echo'. |
| verbose | if 'TRUE', more diagnostics (than just 'echo = TRUE') are printed during parsing and evaluation of input, including extra info for each expression. |
| prompt.echo | character; gives the prompt to be used if 'echo = TRUE'. |
| max.deparse.length | integer; is used only if 'echo' is 'TRUE' and gives the maximal length of the "echo" of a single expression. |
| chdir | logical; if 'TRUE', the R working directory is changed to the directory containing 'file' for evaluating |
| prompt | should a prompt be printed at the end of the evaluation return? |

## Details

This function is usually called by functions that processes commands send by GUI clients.

## Value

The formatted output is returned invisibly.

## Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>), after code written by Tom Short

## See Also

source

# Index