Package 'svMisc'

July 2, 2014

Type Package

Version 0.9-70

Date 2014-03-01

Title SciViews GUI API - Miscellaneous functions

Author Philippe Grosjean [aut, cre],Romain Francois [ctb],Kamil Barton [ctb]

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Imports utils, methods, tools

Depends R (>= 2.13.0)

Suggests svUnit, rJava

Description Supporting functions for the GUI API (various utility functions)

License GPL-2

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

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

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-02 12:40:51

R topics documented:

`svMisc-package` *SciViews GUI API - Miscellaneous functions*

Description

The SciViews svMisc package collects together a series of functions that are shared across all the svXXX packages.

Details

Package:	svMisc
Type:	Package
Version:	0.9-70
Date:	2014-03-01
License:	GPL 2 or above, at your convenience

Author(s)

Philippe Grosjean, Romain Francois & Kamil Barton.

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

`addItems` *Add items from one vector to another one with or without replacement*

Description

The function takes the (named) items of one vector and place them in a second vector if these names do not exist yet there, or replace corresponding content otherwise.

Usage

```
addItems(x, y, use.names = TRUE, replace = TRUE)
addActions(obj = ".svActions", text = NULL, code = NULL, state = NULL,
options = NULL, replace = TRUE)
addIcons(obj = ".svIcons", icons, replace = TRUE)
addMethods(methods)
```

Arguments

`x`	the vector to add items to.
`y`	the vector of which we want to inject missing items in 'x'.
`use.names`	use names of items to determine which one is unique, otherwise, the selection is done on the items themselves.
`replace`	do we replace existing items in 'x'?
`obj`	the name of the object in `SciViews:TempEnv` to manipulate.
`text`	the text of actions to add (label on first line, tip on other lines).
`code`	the R code of actions to add.

state	the default (strating) state of an action, as a succession of letters: \"c\" = checked, \"u\" = unchecked (default); \"d\" = disabled, \"e\" = enabled (default); \"h\" = hidden, \"v\" = visible (default). Default values are facultative. Ex: udv means: unchecked - disabled - visible and it equals to simply d.
options	a character vector with other options to pass to the graphical toolkit for this action.
icons	a named character vector matching names of actions/panels with the URL or file name of icon resources accessible by the GUI client.
methods	the methods to add to getOption("svGUI.methods"). This information is used to compute a list of possible methods for a given object, for instance, in the context menu of an object explorer (see objMenu()).

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

listMethods, objMenu, TempEnv

Examples

```
## I have a vector v1 with this:
v1 <- c(a = "some v1 text", b = "another v1 text")
## I want to add items whose name is missing in v1 from v2
v2 <- c(a = "v2 text", c = "the missign item")
addItems(v1, v2, replace = FALSE)
## Not the same as
addItems(v1, v2, replace = TRUE)
## This yield different result (names not used and lost!)
addItems(v1, v2, use.names = FALSE)

## This is useful to add actions, icons, descriptions, shortcuts or methods
## TODO: examples and use for functions addActions(), addIcons() and
## addMethods()
```

---

addTemp *Add data to an item in a temporary list variable*

Description

The function adds data to an item in a list variable located in SciViews:TempEnv, an environment dedicated to temporary variables (especially useful for GUIs).

Usage

```
addTemp(x, item, value, use.names = TRUE, replace = TRUE)
```

Arguments

x	the name of the variable containing the list.
item	the item to add data to in the list.
value	the value to add in the item, it must be a named vector and element matching is done according to name of items.
use.names	do we match items in the existing vector and the vector we add by means of its names or its values?
replace	do we replace existing items?

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

TempEnv, assignTemp, getTemp, existsTemp, rmTemp, changeTemp, tempvar

Examples

```
addTemp("tst", "item1", c(a = 1, b = 2))
## Retrieve this variable
getTemp("tst")
## Add to item1 in this list without replacement
addTemp("tst", "item1", c(a = 45, c = 3), replace = FALSE)
getTemp("tst")
## Same but with replacement of existing items
addTemp("tst", "item1", c(a = 45, c = 3), replace = TRUE)
getTemp("tst")
## Delete the whole variable
rmTemp("tst")
```

---

argsTip *Show function arguments in a human-readable way - get a call tip*

Description

argsTip() displays function arguments in a better way than args() does. It is primarily intended for code tips in GUIs.

Usage

```
argsTip(name, only.args = FALSE, width = getOption("width"))
callTip(code, only.args = FALSE, location = FALSE, description = FALSE,
    methods = FALSE, width = getOption("width"))
```

Arguments

name	a string with the name of a function.
code	a fraction of R code ending with the name of a function, eventually followed by '('.
only.args	do we return only arguments of the function (arg1, arg2 = TRUE, ...), or the full call, like (myfun(arg1, arg2 = TRUE, ...)).
width	reformat the tip to fit that width, except if width = NULL.
location	if TRUE then the location (in which package the function resides) is appended to the calltip between square brackets.
description	if TRUE then a short description of the function is added to the callTip (in fact, the title of the corresponding help page, if it exists).
methods	if TRUE then a short message indicating if this is a generic function and that lists, in this case, available methods.

Value

A string with the calling syntax of the function, plus additional information, depending on the flag used. Note that methods can considerably slow down the execution, especially for generic functions that have many methods like print(), or summary!

Note

argsTip() is supposed to display S3 and S4 methods, and primitives adequately,... but this is not implemented yet in the current version!

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

args, argsAnywhere

Examples

```
argsTip("ls")
callTip("myvar <- lm(")
```

assignTemp — *Assign a temporary variable in the SciViews:TempEnv environment*

Description

The function assigns a variable to SciViews:TempEnv, an environment dedicated to temporary variables (especially useful for GUIs).

Usage

```
assignTemp(x, value, replace.existing = TRUE)
```

Arguments

x	the name of the variable.
value	the value of the variable.
replace.existing	do we replace an existing variable?

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

TempEnv, changeTemp, getTemp, rmTemp, existsTemp, addTemp, tempvar

Examples

```
assignTemp("test", 1:10)
## Retrieve this variable
getTemp("test")
```

batch — *Run a function in batch mode*

Description

A function can be run in batch mode if it never fails (replace errors by warnings) and return TRUE in case of success, or FALSE otherwise.

Usage

```
batch(items, fun, ..., show.progress = !isAqua() && !isJGR(),
    suppress.messages = show.progress, verbose = TRUE)
```

Arguments

items	the items (usually, arguments vector of character strings) on which to apply fun sequentially.
fun	the function to run (must return TRUE or FALSE and issue only warnings and messages to be a good candidate, batchable, function).
...	further arguments to pass the fun.
show.progress	do we show progression as item x on y... message? This uses the progress() function.
suppress.messages	are messages from the batcheable function suppressed? Only warnings will be issued. Recommended if show.progress = TRUE.
verbose	display start and end messages if TRUE (default).

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

progress

Examples

```
## Here is a fake batcheable process
fakeProc <- function (file) {
    message("Processing ", file, "...")
    flush.console()
    Sys.sleep(0.5)
    if (runif(1) > 0.7) { # Fails
        warning("fakeProc was unable to process ", file)
        return(invisible(FALSE))
    } else return(invisible(TRUE))
}

## Run it in batch mode on five items
files <- paste("file", 1:5, sep = "")
batch(files, fakeProc)
```

---

captureAll — *Run an R expression and capture output and messages in a similar way as it would be done at the command line*

Description

This function captures results of evaluating one or several R expressions the same way as it would be issued at the prompt in a R console. The result is returned in a character string. Errors, warnings and other conditions are treated as usual, including the delayed display of the warnings if options(warn = 0).

Usage

```
captureAll(expr, split = TRUE, echo = TRUE, file = NULL, markStdErr = FALSE)
```

Arguments

expr	a valid R expression to evaluate (names and calls are also accepted).
split	do we split output, that is, do we also issue it at the R console too, or do we only capture it silently?
echo	do we echo each expression in front of the results (like in the console)? In case the expression spans on more than 7 lines, only first and last three lines are echoed, separated by [...].
file	a file, or a valid opened connection where output is sinked. It is closed at the end, and the function returns NULL in this case. If file = NULL (by default), a textConnection() captures the output and it is returned is a character string by the function.
markStdErr	if TRUE, stderr is separated from sddout by STX/ETX character

Value

Returns a string with the result of the evaluation done in the user workspace.

Note

If the expression is provided as a character string that should be evaluated, and you need a similar behaviour as at the prompt for incomplete lines of code (that is, to prompt for more), you should not parse the expression with `parse(text = "<mycode>")` because it returns an error instead of an indication of an incomplete code line. Use `parseText("<mycode>")` instead, like in the examples bellow. Of course, you have to deal with incomplete line management in your GUI/CLI application... the function only returns NA instead of a character string.

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

parseText, parse, expression, capture.output, sourceClipboard

Examples

```
writeLines(captureAll(expression(1+1), split = FALSE))
writeLines(captureAll(expression(1+1), split = TRUE))
writeLines(captureAll(parseText("search()"), split = FALSE))

## Not run:
writeLines(captureAll(parseText('1:2 + 1:3'), split = FALSE))
writeLines(captureAll(parseText("badname"), split = FALSE))

## End(Not run)
```

```
## Management of incomplete lines
captRes <- captureAll(parseText("1 +")) # Clearly an incomplete command
if (is.na(captRes)) cat("Incomplete line!\n") else writeLines(captRes)
rm(captRes)
```

---

changeTemp *Change an item in a temporary list variable*

Description

The function changes an item in a list variable located in SciViews:TempEnv, an environment dedicated to temporary variables (especially useful for GUIs).

Usage

```
changeTemp(x, item, value, replace.existing = TRUE)
```

Arguments

x	the name of the variable containing the list.
item	the item to change (or create) in the list.
value	the value to put in the list item.
replace.existing	do we replace an existing item?

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

TempEnv, assignTemp, getTemp, existsTemp, rmTemp, addTemp, tempvar

Examples

```
changeTemp("tst", "item1", 1:10)
## Retrieve this variable
getTemp("tst")
## Create another item in the list
changeTemp("tst", "item2", TRUE)
getTemp("tst")
## Change it
changeTemp("tst", "item2", FALSE)
getTemp("tst")
## Delete it (= assign NULL to it)
changeTemp("tst", "item2", NULL)
getTemp("tst")
## Delete the whole variable
rmTemp("tst")
```

compareRVersion *Compare current R version with a specified one*

Description

Determine if R is older (return -1), or not (return 0 if equal, or 1 if newer) than a given version number.

Usage

```
compareRVersion(version)
```

Arguments

version	a string defining the version to compare to, like '2.0.0' or '1.9.1'.

Value

-1 if R is older, 0 if equal, 1 if newer. Take care: if you specify version as "2.11", and R is version "2.11.0", then the function will return 1 (newer)!

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

compareVersion, R.version, isAqua, isWin

Examples

```
compareRVersion("2.11.0") # Note that we strongly advise to use R > 2.11.0!
```

completion *Get a completion list for a R code fragment*

Description

Returns names of objects/arguments/namespaces matching a code fragment.

Usage

```
completion(code, pos = nchar(code), min.length = 2, print = FALSE,
    types = c("default", "scintilla"), addition = FALSE, sort = TRUE,
    what = c("arguments", "functions", "packages"), description = FALSE,
    max.fun = 100, skip.used.args = TRUE, sep = "\n", field.sep = "\t")
```

Arguments

code	a partial R code to be completed.
pos	the position of the cursor in this code.
min.length	the minimal length in characters of code required before the completion list is calculated.
print	logical, print result and return invisibly. See details.
types	a named list giving names of types. Set to NA to give only names. See details.
addition	should only addition string be returned?
sort	do wer sort the list of completions alphabetically?
what	what are we looking for? Allow to restrict search for faster calculation.
description	do we describe items in the completion list (may be slow)?
max.fun	in case we describe items, the maximum number of functions to process (if longer, no description is returned for function) because it can bz very slow otherwise.
skip.used.args	logical, if completion is within function arguments, should the already used named arguments be omitted?
sep	the separator to use between returned items.
field.sep	character string to separate fields for each entry.

Details

The completion list is context-dependent, and it is calculated as if the code was entered at the command line.

If the code ends with $ or [[, then the function look for items in a list or data.frame whose name is the last identifier.

If the code ends with @, then the function look for slots of the corresponding S4 object.

If the code ends with ::, then it looks for objects in a namespace.

If the code ends with a partial identifier name, the function returns all matching keywords visible from .GlobalEnv.

If the code is empty or parses into an empty last token, the list of objects currently in the global environment is returned.

Take care: depending on the context, the completion list could be incorrect (but it should work for code entered at the command line). For instance, inside a function call, the context is very different, and arguments and local variables should be returned instead. This may be implemented in the future, but for now, we focus on completion that should be most useful for novice useRs that are using rather simple commands entered one after the other in a script (and considering the script is run or sourced line after line in R).

There are other situations where the completion can be calculated, see the help of rc.settings().

If print == TRUE, results are returned invisibly, and printed in a form: triggerPos[newline]completions separated by sep.

If types are supplied, a completion will consist of name and type, separated by type.sep. types may me a vector of length 5, giving the type codes for "function", "variable", "environment", "argument" and "keyword". If types == "default", above type names are given; types == "scintilla" will give numeric codes that can be used with "scintilla.autoCShow" function (SciViews-K Komodo Edit plugin).

Value

If types == NA and description = FALSE, a character vector giving the completions, otherwise a data frame with two columns: 'completion', and 'type' when description = FALSE, or with four columns: "completion', 'type', 'desc' and 'context' when description = TRUE.
Attributes:
attr(,"token") - a completed token.
attr(,"triggerPos") - number of already typed characters.
attr(,"fguess") - name of guessed function.
attr(,"isFirstArg") - is this a first argument?

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org> & Kamil Barton

See Also

rc.settings.

Examples

```
## a data.frame
data(iris)
completion("item <- iris$")
completion("item <- iris[[")

## An S4 object
setClass("track", representation(x = "numeric", y = "numeric"))
t1 <- new("track", x = 1:20, y = (1:20)^2)
completion("item2 <- t1@")

## A namespace
completion("utils::", description = TRUE)

## A partial identifier
completion("item3 <- va", description = TRUE)

## Otherwise, a list with the content of .GlobalEnv
completion("item4 <- ")

## TODO: directory and filename completion!

rm(iris, t1)
```

`def` *Define a vector of a given mode and length (possibly filling it with default values)*

Description

This function makes sure that a vector of a given mode and length is returned. If the value provided is `NULL`, or empty, the default value is used instead. If `length.out = NULL`, the length of the vector is not constrained, otherwise, it is fixed (possibly cutting or recycling `value`).

Usage

```
def(value, default = "", mode = "character", length.out = NULL)
```

Arguments

`value`	the value to pass with default.
`default`	the default value to use, in case of `NULL`, or `length(value) == 0`.
`mode`	the mode of the resulting object: 'character', 'logical', 'numeric' (and, if you want to be more precise: 'double', 'integer' or 'single') or 'complex'. Although not being a mode by itself, you can also specify 'factor' to make sure the result is a factor (thus, of mode 'numeric', storage mode 'integer', with a levels attribute). Other modes are ignored, and `value` is NOT coerced (silently) in this case, i.e., if you don't want to force coercion of the resulting object, use anything else.
`length.out`	the desired length of the returned vector; use `length.out = NULL` (default) if you don't want to change the length of the vector.

Value

A vector of given mode and length, with either `value` or `default`.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`mode`, `rep` `getTemp`, `assignTemp`

Examples

```
def(1:3, length.out = 5)                  # Convert into character and recycle
def(0:2, mode = "logical")                   # Numbers to logical
def(c("TRUE", "FALSE"), mode = "logical")    # Text to logical
def(NULL, "default text")                    # Default value used
def(character(0), "default text")            # Idem
def(NA, 10, mode = "numeric", length.out = 2) # Vector of two numbers
```

descFun *Get textual help on function or function arguments*

Description

Textual help on functions or their arguments is extracted for text online help for a given function. By default, all arguments from the online help are returned for `descArgs()`. If the file contains help for several functions, one probably gets also some irrelevant information. Use of 'args' to limit result is strongly encouraged.

Usage

```
descFun(fun, package, lib.loc = NULL)
descArgs(fun, args = NULL, package = NULL, lib.loc = NULL)
```

Arguments

`fun`	a character string with the name of a function (several functions accepted for `descFun()`.
`args`	either `NULL` (by default) to return the description of all arguments from the corresponding man page, or a character vector with names of the arguments to search for.
`package`	a character string with the name of the package that contains `fun`, or `NULL` for searching in all loaded packages.
`lib.loc`	a character vector of directory names of R libraries, or `NULL`. The default value of `NULL` corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.

Value

A string with the description of the function or of its arguments. If the man page is not found, a vector of empty strings is returned. Empty strings are also returned for arguments that are not found in the man page.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`completion`

Examples

```
descFun("ls", "base")
descFun("library", "base")
descFun("descFun", "svMisc")
descFun("descArgs")
descArgs("ls")
descArgs("library", args = c("package", "pos"))
```

---

existsTemp *Determine if a variable exists in SciViews:TempEnv*

Description

Does a variable exist in the SciViews:TempEnv environment?

Usage

```
existsTemp(x, mode = "any")
```

Arguments

x	the name of the variable (character string).
mode	the mode of the seeked variable.

Value

TRUE if the variable exists in SciViews:TempEnv (and is of the correct mode), FALSE otherwise.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

TempEnv, assignTemp, changeTemp, getTemp, rmTemp, addTemp

Examples

```
assignTemp("test", 1:10)
## Check if this variable exists
existsTemp("test")
## Remove it
rmTemp("test")
## Does it still exist?
existsTemp("test")
```

`fileEdit` *Invoke an external text editor for a file*

Description

Edit a text file using an external editor. Possibly wait for the end of the program and care about creating the file (from a template) if it does not exists yet.

Usage

```
fileEdit(..., title = files, editor = getOption("fileEditor"), file.encoding = "",
    template = NULL, replace = FALSE, wait = FALSE)
```

Arguments

`...`	path to one or more files to edit.
`title`	the title of the editor window (not honoured by all editors, most external editors only display the file name or path).
`editor`	editor to use. Either the name of the program, or a string containing the command to run, using %s as replacement tag where to place the filename in the command, or a function with 'file', 'title' and 'wait' arguments to delegate process of the files.
`file.encoding`	encoding of the files. If `""` or `native.enc`, the files are considered as being already in the right encoding.
`template`	one or more files to use as template if files must be created. If `NULL`, an empty file is created. This argument is recycled for all files to edit.
`replace`	force replacement of files if `template` is not null.
`wait`	wait for edition to complete. If more than one file is edited, the program waits sequentially for each file to be edited in turn (with a message in the R console).

Value

The function returns `TRUE` if it was able to edit the files or `FALSE` otherwise, invisibly. Encountered errors are reported as warnings.

Note

The default editor program, or the command to run is in the `fileEditor` option (use `getOption("fileEditor")` to retrieve it, and `options(fileEditor = "<myowneditor>")` to change it). Default values are determined automatically.

On Unixes, "gedit", "kate" and "vi" are looked for in that order. Note that there is a gedit plugin to submit code directly to R: http://rgedit.sourceforge.net/. Since, gedit natively supports a lot of different syntax highlighting, including R, and is lightweight but feature rich, it is recommended as default text editor for `fileEdit()` on Unixes. If JGR is run and the editor is "vi" or "internal", then the internal JGR editor is used, otherwise, the provided editor is chosen.

On Mac OS X, if the "edit" program exists, it is used (it is the command line program installed by TextWrangler or BBEdit, see http://www.barebones.com/products/, much more capables text editors than the default TextEdit program), otherwise, the default text editor used by OS X is choosen (default usually to TextEdit). TextWrangler can be installed freely. It can be configured to highlight and submit R code. see http://macsci.jelmerborst.nl/files/textwrangler_and_r.php for instructions. It features also several tools that makes it a much better choice than TextEdit for `fileEdit()` on Mac OS X. Specify "textwrangler" or "bbedit" to force using these programs. The default value is "textedit", the Mac default text editor, but on R.app, and with `wait = FALSE`, the internal R.app editor is used instead in that case. If JGR is run, and the editor is "textedit", "internal" or "vi", then, the internal JGR editor is used instead.

On Windows, if Notepad++ is installed in its default location, it is used, otherwise, the default "notepad" is used in Rterm and the internal editors are chosen for Rgui. Notepad++ is a free text editor that is much better suited to edit code or text files that the default Windows' notepad application, in particular because it can handle various line end types (Unix, Mac or Windows) and encodings. It also supports syntax highlighting, code completion and much more. So, it is strongly recommended to install it (see http://notepad-plus-plus.org/) and use it with `fileEdit()`. There is also a plugin to submit code to R directly from Notepad++: http://sourceforge.net/projects/npptor/.

Of course, you can use your own text editor, just indicate it in the `fileEditor` option. Note, however, that you should use only lighweight and fast starting programs. Also, for the `wait = TRUE` argument of `fileEdit()`, you must check that R waits for the editor to be closed before further processing code. In some cases, a little command line program is used to start a larger application (like for Komodo Edit/IDE), or the program delegates to an existing instances and exits immediatelly, even if the file is still edited. Such editors are not recommended at all for `fileEdit()`.

If you want to use files that are compatibles between all platforms supported by R itself, you should think about using ASCII encoding as much as possible and the Windows style of line-ending. That way, you ensure that all the default editors will handle those files correctly, including the broken default editor on Windows, notepad, which does not understand at all Mac or Unix line ends!

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`systemFile`, `file.path`, `file.exists`, `file.edit`

Examples

```
## Not run:
## Create a template file in the tempdir...
tpl <- tempfile("template", fileext = ".txt")
cat("Example template file\nto be used with fileEdit()\n", file = tpl)

## ... and edit a new file, starting from that template:
newf <- tempfile("test", fileext = ".txt")
fileEdit(newf, template = tpl, wait = TRUE)

## Make sure the content ends with \n, and read it
```

```
cat("\n", file = newf, append = TRUE)
cat("Your file contains:\n")
readLines(newf)

## Eliminate both the file and template
unlink(newf)
unlink(tpl)

## End(Not run)
```

---

getTemp *Get a temporary variable from the SciViews:TempEnv environment*

Description

The function gets a variable, or an item in a list variable from SciViews:TempEnv, an environment dedicated to temporary variables.

Usage

```
getTemp(x, default = NULL, mode = "any", item = NULL)
```

Arguments

x	the name of the variable.
default	the default value to return, in case the variable or the item does not exist.
mode	the mode of the variable or the item (if the variable exists, but is not of correct mode, the default value is returned). Use `mode = "any"` (default value) to retrieve the variable or item whatever its mode.
item	if `NULL` (default), the whole variable content is retrieve, otherwise, the variable is considered as a list, and the corresponding item from that list is returned. In this case, `default` and `mode` arguments correspond to the item, not to the whole variable.

Value

The content of the variable, of the item, or the default value if the variable or item is not found in SciViews:TempEnv, or of the wrong mode.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

TempEnv, assignTemp, changeTemp, rmTemp, existsTemp, addTemp

Examples

```
assignTemp("test", 1:10)
## Retrieve this variable
getTemp("test")
## Retrieve a non existing variable (returns default value)
getTemp("nonexistant", default = "default value")
## Set and retrieve items from a list
changeTemp("test2", "item1", 1:5)
getTemp("test2", item = "item1")
## Compare to:
getTemp("test2")
## Remove temporary variables
rmTemp(c("test", "test2"))
```

---

guiCmd *Execute a command in the GUI client*

Description

This function is not intended to be used at the command line (except for debugging purposes). It executes a command string to a (compatible) GUI client.

Usage

```
guiCmd(command, ...)
guiImport(...)
guiExport(...)
guiLoad(...)
guiReport(...)
guiSave(...)
guiSetwd(...)
guiSource(...)
```

Arguments

command	the command string to execute in the GUI client.
...	parameters provided for the command to execute in the GUI client.

Details

You must define a function `.guiCmd()` in the `SciViews:TempEnv` environment that will take first argument as the name of the command to execute (like `source`, `save`, `import`, etc), and ... with arguments to the command to send. Depending on your GUI, you should have code that delegates the GUI elements (ex: display a dialog asking for a .Rdata file to source) and then, execute the command in R with the selected file as attribute.

Value

The result of the command if it succeed, or NULL if the command cannot be run (i.e., .guiCmd() is not defined in SciViews:TempEnv).

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

assignTemp

---

helpSearchWeb *Search web documents about R and R functions*

Description

Retrieve web documents, messages in R mailing lists, or wiki containing apropos string.

Usage

```
helpSearchWeb(what, type = c("R", "archive", "wiki", "google"), browse = TRUE,
    msg = browse, ...)
```

Arguments

what	the string(s) to search. In case of several strings, or several words, any of these words are searched.
type	the search engine, or location to use.
browse	do we actually show the page in the Web browser? If type = "R", this argument is ignored and the result is always displayed in the Web browser.
msg	do we issue a message indicating that a page should be displayed shortly in the Web browser? If type = "R", this argument is ignored and a message is always displayed.
...	further arguments to format the result page in case of type = "R". These are the same arguments as for RSiteSearch().

Value

Returns the URL used invisibly (invoked for its side effect of opening the Web browser with the search result, when browse = TRUE).

Note

The RSiteSearch() function in the 'utils' package is used when type = "R".

Author(s)

David Forrest <drf@vims.edu> & Philippe Grosjean <phgrosjean@sciviews.org> after Barry Rowland's original code

See Also

RSiteSearch, help.search

Examples

```
## Not run:
helpSearchWeb("volatility")                    # R site search, by default
helpSearchWeb("volatility", type = "google")   # Google search
helpSearchWeb("volatility", type = "archive")  # In the mailing list archive
helpSearchWeb("median mean", type = "wiki")    # In the R Wiki

## End(Not run)
```

---

isAqua *Is R running with the AQUA GUI (R.app or R64.app)?*

Description

Determine if the R UI is AQUA, the standard R GUI under Macintosh. This function can also be used under other platforms, but it will always return FALSE.

Usage

```
isAqua()
```

Value

TRUE if the R UI is AQUA, FALSE otherwise.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

isMac, isRgui, isJGR

Examples

```
isAqua()
```

`isHelp` *Is there a help file and example to run associated with an object?*

Description

Determine if 'topic' has a help file and example to run.

Usage

```
isHelp(topic, package = NULL, lib.loc = NULL)
```

Arguments

`topic`	name or literal character string: the online help topic to look for.
`package`	a character vector giving the package names to look into for help or example code, or `NULL`. By default, all packages in the search path are used.
`lib.loc`	a character vector of directory names of R libraries, or `NULL`. The default value of `NULL` corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries.

Value

A logical vector with two elements. The first one indicating if there is a help file, and the second one indicating if there are examples associated with this help file.

Note

This code is largely inspired from the first part of `example()`.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

Examples

```
isHelp("help") # Help and example
isHelp("Rtangle") # Help but no example
isHelp("notopic") # No help or example
```

isJGR *Is R running with the JGR GUI?*

Description

Determine if the R GUI is JGR.

Usage

```
isJGR()
```

Value

TRUE if the R GUI is JGR, FALSE otherwise.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

isMac, isAqua, isRgui

Examples

```
isJGR()
```

isMac *Is it a Macintosh platform?*

Description

This is a shortcut function to determine if R is currently running on a Macintosh platform.

Usage

```
isMac()
```

Value

TRUE if the platform is Macintosh, FALSE otherwise.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

isWin, isAqua, isJGR

Examples

```
isMac()
```

---

isRgui *Is Rgui.exe (under Windows) running?*

Description

Determine if the R UI is Rgui, the standard R program under Windows. This function can also be used under other platforms, but it will always return FALSE.

Usage

```
isRgui()
```

Value

TRUE if the R UI is Rgui.exe, FALSE otherwise.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

isSDI, isWin, isJGR

Examples

```
isRgui()
```

isSDI *Is Rgui started in SDI mode under Windows?*

Description

Determine if R is running in Rgui under Windows and is in SDI mode.

Usage

```
isSDI()
```

Value

`TRUE` if it is Rgui under Windows, and it is stared in SDI mode (Single-Document Interface), `FALSE` otherwise (either in Rgui in MDI mode, or another UI).

Note

See the menu entry 'Edit' -> 'GUI preferences' to change the Rgui mode, or start Rgui with the '–SDI' argument line parameter. Under another platform than Windows or if it is not Rgui, then `isSDI()` always returns `FALSE`

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org> & Eric Lecoutre

See Also

`isRgui`, `isWin`

Examples

```
isSDI()
```

isWin *Is it a Windows platform?*

Description

Shortcut to determine if the platform is Windows.

Usage

```
isWin()
```

Value

TRUE if the platform is Windows, FALSE otherwise.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

isMac, isRgui, isJGR

Examples

```
isWin()
```

---

listMethods *List all methods associated with a generic function or a class*

Description

List all S3 and/or S4 methods for a generic function or for a class.

Usage

```
listMethods(f = character(), class = NULL, S3 = TRUE, S4 = TRUE,
    mixed = TRUE, filter = getOption("svGUI.methods"))
```

Arguments

f	the name of the generic function (character string), used only when class = NULL.
class	the name of a class.
S3	if TRUE, list of S3 methods.
S4	if TRUE, list of S4 methods.
mixed	if TRUE, S3 and S4 methods are mixed together in a character vector, otherwise, S3 and S4 methods are reported separately in a list
filter	a list of methods to consider when listing class methods. Only classes in this list that are defined for the class are returned. Store the list of methods you want in the options \"svGUI.methods\". The package proposes a reasonable starting point on loading if this option is not defined yet.

Value

If mixed = TRUE, a list with components:

S3	The S3 methods for the generic function or the class, or character(0) if none
S4	The S4 methods for the generic function or the class, or character(0) if none

Otherwise, a character vector with the requested methods

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

listTypes, addItems

Examples

```
## Generic functions
listMethods("t.test")               # S3
listMethods("show", mixed = FALSE) # S4
listMethods("ls") # None, not a generic function!

## Classes
## Only the following methods are considered
getOption("gui.methods")
listMethods(class = "data.frame")
listMethods(class = "lm")
```

---

listTypes *List all types for a method*

Description

List all types for a method; types are variants for a given method defined in a way it is easy to add other variants dynamically (on the contrary to a usual `type =` or `which =` argument, like in `plot.ts()` or `plot.lm()`, respectively.

Usage

```
listTypes(method, class = "default", strict = FALSE)
```

Arguments

method	the method name.
class	the class name.
strict	do we list only types for the class (TRUE), or all possible types, including for inherited objects, and default ones (FALSE, by default)?

Value

A vector with character strings with methods' type names.

Note

This function is only useful for special generic functions with type argument like `view`, `copy` or `export`. These functions offer a mechanism to easily add custom types, and the present function list them. For S3 objects a type is simply a function whose name is : <method>_<type>.<class>. So, adding new <type>s for <method> is very easy to implement.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`listMethods`

Examples

```
listTypes("view")  # All default view types currently defined
listTypes("view", "data.frame")
listTypes("view", "data.frame", TRUE) # None, except if you defined custom views!
```

---

objBrowse *Functions to implement an object browser*

Description

These functions provide features required to implement a complete object browser in a GUI client.

Usage

```
objBrowse(id = "default", envir = .GlobalEnv, all.names = NULL, pattern = NULL,
    group = NULL, sep = "\t", path = NULL, regenerate = FALSE)
objClear(id = "default")
objDir()
objInfo(id = "default", envir = .GlobalEnv, object = "", path = NULL)
objList(id = "default", envir = .GlobalEnv, object = NULL, all.names = FALSE,
    pattern = "", group = "", all.info = FALSE, sep = "\t", path = NULL,
    compare = TRUE, ...)

objMenu(id = "default", envir = .GlobalEnv, objects = "", sep = "\t",
    path = NULL)
objSearch(sep = "\t", path = NULL, compare = TRUE)

## S3 method for class 'objList'
print(x, sep = NA, eol = "\n", header = !attr(x, "all.info"),
    raw.output = !is.na(sep), ...)

write.objList(x, path, sep = "\t", ...)
```

Arguments

id	the id of the object browser (you can run several ones concurrently, providing you give them different ids).
envir	an environment, or the name of the environment, or the position in the search() path.
all.names	do we display all names (including hidden variables starting with '.')?
pattern	a pattern to match for selecting variables.
group	a group to filter.
path	the path where to write a temporary file with the requested information. Set to NULL (default) if you don't pass this data to your GUI client by mean of a file.
regenerate	do we force to regenerate the information?
object	name of the object selected in the object browser, components/arguments of which should be listed.
objects	a list with selected items in the object browser
all.info	do we return all the information (envir as first column or not (by default).
compare	if TRUE, result is compared with last cached value and the client is updated only if something changed.
sep	separator to use between items (if path is not NULL).
x	object returned by objList.
eol	separator to use between object entries, default is to list each item in a separate line.
header	if TRUE, two-line header is printed, of the form: Environment = environment name Object = object name Default is not to print header if all.info is true.
raw.output	if TRUE, a compact, better suited for parsing output is produced.
...	further arguments, passed to write.table.

Details

objBrowse() does the horsework. objDir() gets the temporary directory where exchange files are stored, in case you exchange data through files. You can use a better way to communicate with your GUI (you have to provide your code) and disable writing to files by using path = NULL.

objList() lists objects in a given environment, elements of a recursive object or function argument.

objSearch() lists the search path.

objClear() clears any reference to a given object browser.

objInfo() computes a tooltip info for a given object.

objMenu() computes a context menu for selected object(s) in the object explorer managed by the GUI client.

print.objList() print method for objList objects.

Value

Depending on the function, a list, a string, a reference to an external, temporary file or TRUE in case of success or FALSE otherwise is returned invisibly.

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>) and Kamil Barton (<kbarton@zbs.bialowieza.pl>)

See Also

completion, callTip

Examples

```
## Create various context menus
data(iris)
(objInfo(object = "iris"))
data(trees)
## For one object
(objMenu(objects = "iris"))
## For multiple objects
(objMenu(objects = c("iris", "trees")))
## For inxistant object (return "")
(objInfo(object = "noobject"))
(objMenu(objects = "noobject"))
rm(iris, trees)

## For environments
(objInfo(envir = ".GlobalEnv"))
(objMenu(envir = ".GlobalEnv"))
(objInfo(envir = "SciViews:TempEnv"))
(objMenu(envir = "SciViews:TempEnv"))
(objInfo(envir = "package:datasets"))
(objMenu(envir = "package:datasets"))
## For an environment that does not exist on the search path (return "")
(objInfo(envir = "noenvir"))
(objMenu(envir = "noenvir"))
```

---

package *A very silent and multipackage require() function*

Description

This function loads one or several R packages as silently as possible and it returns TRUE only if all packages are loaded successfully. If at least one loading fails, a short message is printed.

Usage

```
package(..., warn = TRUE)
```

Arguments

...	the name of one or several R packages to load (character strings).
warn	If TRUE, issue a warning if one or several packages are not loaded.

Value

TRUE if all packages are loaded correctly, FALSE otherwise. This function is designed to concisely and quitely indicate package requirements in GUI menu or other GUI actions.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

require, pkgManLoad

Examples

```
## This should work...
if (all(package("tools", "methods"))) cat("Fine!\n")
## ... but this not
if (!all(package("tools", "badname", warn = FALSE))) cat("Not fine!\n")
```

---

parseText *Parse a character string as if it was a command entered at the command line*

Description

Parse R instructions provided as a string and return the expression if it is correct, or an object of class 'try-error' if it is an incorrect code, or NA if the (last) instruction is incomplete.

Usage

```
parseText(text, firstline = 1, srcfilename = NULL, encoding = "unknown")
```

Arguments

text	the character string vector to parse into an R expression.
firstline	the index of first line being parsed in the file. If this is larger than 1, empty lines are added in front of text in order to match the correct position in the file.
srcfilename	a character string with the name of the source file.
encoding	encoding of text, as in parse.

Value

Returns an expression with the parsed code or NA if the last instruction is correct but incomplete, or an object of class'try-error' with the error message if the code is incorrect.

Note

On the contrary to `parse()`, `parseText()` recovers from incorrect code and also detects incomplete code. It is also easier to use in case you pass a character string to it, because you don't have to name the argument explicitly (`text = ...`).

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

See Also

`captureAll`, `sourceClipboard`, `parse`

Examples

```
parseText('1+1')
parseText('1+1; log(10)')
parseText(c('1+1', 'log(10)'))

## Incomplete instruction
parseText('log(')

## Incomplete strings
parseText('text <- "some string')
parseText("text <- 'some string")

## Incomplete names (don't write backtick quoted names on several lines!)
## ...but just in case
parseText('`myvar')

## Wrong instruction
parseText('log)')
```

---

pkgMan *Functions to manage R side of the SciViews R package manager*

Description

These functions should not be used directly by the end-user. They implement the R-side code for the SciViews R package manager.

Usage

```
pkgManDescribe(pkgName, print.it = TRUE)
pkgManInstall(pkgs, install.deps = FALSE, ask = TRUE)
pkgManGetInstalled(sep = ";", eol = "\t\n")
pkgManGetAvailable(page = "next", pattern = "", n = 50,
    keep = c("Package", "Version", "InstalledVersion", "Status"),
    reload = FALSE, sep = ";", eol = "\t\n")
pkgManLoad(pkgName)
pkgManDetach(pkgName)
pkgManRemove(pkgName)
pkgManGetMirrors()
pkgManSetCRANMirror(url)
```

Arguments

pkgName	the name of one R package (character string).
print.it	Should the result be printed?
pkgs	a list of packages to install.
install.deps	do we also install dependencies?
ask	do we prompt the user for package installation?
sep	field separator to use.
eol	end-of-line sequence to use.
page	which page to get?
pattern	selection pattern.
n	the number of items to retrieve.
keep	the columns to keep in the resulting data frame.
reload	do we force reload of the data and ignore cache version?
url	the URL to use for the current CRAN mirror.

Value

These functions return data that is intended to be used by the SciViews R package manager.

Author(s)

Kamil Barton <kamil.barton@uni-wuerzburg.de>

See Also

package

progress *Display progression of a long calculation on the console and/or in a GUI*

Description

Display progression level of a long-running task in the console. Two mode can be used: either percent of achievement (55%), or the number of items or steps done on a total (1 file on 10 done...). This is displayed either through a message, or through a text-based "progression bar" on the console, or a true progression bar widget in a GUI.

Usage

```
progress(value, max.value = NULL, progress.bar = FALSE, char = "|",
    init = (value == 0), console = TRUE, gui = TRUE)
```

Arguments

value	current value of the progression (use a value higher than max.value to dismiss the progression indication automatically.
max.value	maximum value to be achieved.
progress.bar	should we display a progression bar on the console? If FALSE, a temporary message is used.
char	the character to use to fill the progress bar in the console. not used for the alternate display, or for GUI display of progression.
init	do we have to initialize the progress bar? It is usually done the first time the function is used, and the default value init = (value == 0) is correct most of the time. You must specify init = TRUE in two cases: (1) if the first value to display is different from zero, and (2) if your code issues some text on screen during progression display. Hence, you must force redraw of the progression bar.
console	do we display progression on the console?
gui	do we display progression in a dialog box, or any other GUI widget? See "details" and "examples" hereunder to know how to implement your own GUI progression indicator.

Details

The function progress() proposes different styles of pregression indicators than the standard txtProgressBar in package 'utils'.

The function uses backspace (\8) to erase characters at the console.

With gui = TRUE, the function looks for all functions defined in the '.progress' list located in SciViews:TempEnv environment. Each function is executed in turn with following call: theGUIfunction(value, max.valu
You are responsible to create theGUIfunction() and to add it as an element in the .progress list in SciViews:TempEnv. See also example (5) hereunder.

If your GUI display of the progression offers the possibility to stop calculation (for instance, using a 'Cancel' button), you are responsible to pass this info to your code doing the long calculation and to stop it there. Example (5) shows how to do this.

Value

This function returns NULL invisibly. It is invoked for its side effects.

Note

In a GUI, it is preferable to use a non modal dialog box with a progress widget, or to display such a progress widget in the status bar of your main window.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

batch

Examples

```
## 1) A simple progress indicator in percent
for (i in 0:101) {
progress(i)
Sys.sleep(0.01)
if (i == 101) cat("Done!\n")
}

## 2) A progress indicator with 'x on y'
for (i in 0:31) {
progress(i, 30)
Sys.sleep(0.02)
if (i == 31) cat("Done!\n")
}

## 3) A progress bar in percent
for (i in 0:101) {
progress(i, progress.bar = TRUE)
Sys.sleep(0.01)
if (i == 101) cat("Done!\n")
}

## 4) A progress indicator with 'x on y'
for (i in 0:21) {
progress(i, 20, progress.bar = TRUE)
Sys.sleep(0.03)
if (i == 21) cat("Done!\n")
}

## 5) A progression dialog box with Tcl/Tk
```

```
## Not run:
if (require(tcltk)) {
    guiProgress <- function(value, max.value) {
        ## Do we need to destroy the progression dialog box?
        if (value > max.value) {
            try(tkdestroy(getTemp("guiProgressWindow")), silent = TRUE)
            ## Clean temporary variables
            rmTemp(c("guiProgressState", "guiProgressWindow", "guiProgressCancel"))
            ## ...and exit
            return(invisible(FALSE))
        } else if (existsTemp("guiProgressWindow") &&
            !inherits(try(tkwm.deiconify(tt <- getTemp("guiProgressWindow")),
                silent = TRUE), "try-error")) {
            ## The progression dialog box exists
            ## Focus on it and change progress value
            tkfocus(tt)
            State <- getTemp("guiProgressState")
            tclvalue(State) <- value
        } else {
            ## The progression dialog box must be (re)created
            ## First, make sure there is no remaining "guiProgressStop"
            rmTemp("guiProgressCancel")
            ## Create a Tcl variable to hold current progression state
            State <- tclVar(value)
            assignTemp("guiProgressState", State)
            ## Create the progression dialog box
            tt <- tktoplevel()
            assignTemp("guiProgressWindow", tt)
            tktitle(tt) <- "Waiting..."
            sc <- tkscale(tt, orient = "h", state = "disabled", to = max.value,
                label = "Progress (
            tkpack(sc)
            but <- tkbutton(tt, text = "Cancel", command = function() {
                ## Set a flag telling to stop running calculation
                assignTemp("guiProgressCancel", TRUE) # Content is not important!
                ## Destroy the window
                tkdestroy(tt)
            })
            tkpack(but)
        }
        return(invisible(TRUE))
    }
    ## Register it as function to use in progress()
    changeTemp(".progress", "guiProgress", guiProgress, replace.existing = FALSE)
    rm(guiProgress) # Don't need this any more
    ## Test it...
    for (i in 0:101) {
     progress(i) # Could also set console = FALSE for using the GUI only
     Sys.sleep(0.05)
     ## The code to stop long calc when user presses "Cancel"
     if (existsTemp("guiProgressCancel")) {
         progress(101, console = FALSE) # Make sure to clean up everything
         break
```

```
    }
    if (i == 101) cat("Done!\n")
   }
   ## Unregister the GUI for progress
   changeTemp(".progress", "guiProgress", NULL)
}

## End(Not run)
```

`rmTemp` *Remove one or several temporary variable(s) from the SciViews:TempEnv environment*

Description

The function removes one or more variable(s) from SciViews:TempEnv.

Usage

```
rmTemp(x)
```

Arguments

`x`	the name of the variable (character string), or a vector of characters with the name of all variables to remove from SciViews:TempEnv.

Value

Return `TRUE` if variable existed and is deleted, and `FALSE` otherwise. For multiple variable, a vector of booleans is returned.

Warning

This command issues no error message if variable(s) do not exist in SciViews:TempEnv!

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`TempEnv`, `assignTemp`, `changeTemp`, `getTemp`, `existsTemp`, `addTemp`

Examples

```
assignTemp("test", 1:10)
## Retrieve this variable
getTemp("test")
## Remove it
rmTemp("test")
## Try to retrieve it again
getTemp("test")
```

`sourceClipboard` *Source code from the clipboard*

Description

This function reads R code from the clipboard, and then source it. Clipboard is managed correctly depending on the OS (Windows, Mac OS X, or *nix)

Usage

```
sourceClipboard(primary = TRUE, ...)
```

Arguments

`primary`	only valid on *nix: read the primary (or secondary) clipboard.
`...`	further parameters passed to `source()`.

Value

Same result as `source()`.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`source`, `file`

`systemFile` *Get a system file or directory*

Description

Get system files or directories, in R subdirectories, in package subdirectories, or elsewhere on the disk (including executables that are accessible on the search path).

Usage

```
systemFile(..., exec = FALSE, package = NULL, lib.loc = NULL)
systemDir(..., exec = FALSE, package = NULL, lib.loc = NULL)
```

Arguments

`...`	one or several executables if `exec = TRUE`, or subpath to a file or dir in a package directory if `package != NULL`, or a list of path and subpaths for testing the existence of a file on disk, or a list of directory components to retrieve in 'temp', 'sysTemp', 'user', 'home', 'bin', 'doc', 'etc' and/or 'share' to retrieve special system directories.
`exec`	if `TRUE` (default) search for executables on the search path. It superseedes all other arguments.
`package`	the name of one package to look for files or subdirs in its main directory (use `exec = FALSE` to search inside package dirs).
`lib.loc`	a character vector with path names of R libraries or `NULL` (search all currently known libraries in this case).

Value

A string with the path to the directories or files, or `""` if they are not found, or of the wrong type (a dir for `systemFile()` or or a file for `systemDir()`).

Note

These function aggregate the features of several R functions in package base: system.file(), R.home(), tempdir(), Sys.which(), and aim to provide a unified and convenient single interface to all of them. We make sure also to check that returned components are respectively directories and files for `systemDir()` and `systemFile()`.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`fileEdit`, `file.path`, `file.exists`

Examples

```
systemFile("INDEX", package = "base")
systemFile("help", "AnIndex", package = "splines")
systemFile(package = "base")  # This is a dir, not a file!
systemFile("zip", exec = TRUE)
systemFile("ftp", "ping", "zip", "nonexistingexe", exec = TRUE)
systemDir("temp")             # The R temporary directory
systemDir("sysTemp")          # The system temporary directory
systemDir("user")             # The user directory
systemDir("home", "bin", "doc", "etc", "share")  # Various R dirs
systemDir("zip", exec = TRUE) # Look for the dir of an executable
systemDir("ftp", "ping", "zip", "nonexistingexe", exec = TRUE)
systemDir(package = "base")   # The root of the 'base' package
systemDir(package = "stats")  # The root of package 'stats'
systemDir("INDEX", package = "stats") # This is a file, not a dir!
systemDir("help", package = "splines")
```

TempEnv *Get an environment dedicated to temporary variables (and create it if needed)*

Description

This function creates and returns the reference to a SciViews:TempEnv environment located just before `Autoloads` on the search path.

Usage

```
TempEnv()
```

Value

SciViews:TempEnv (an environment object)

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`assignTemp`, `changeTemp`, `getTemp`, `rmTemp`, `existsTemp`, `addTemp`

Examples

```
TempEnv()
```

tempvar *Get an arbitrary name for a temporary variable*

Description

The function ensures that the variable name is cryptic enough and is not already used.

Usage

```
tempvar(pattern = ".var")
```

Arguments

pattern	the prefix for the variable (the rest is a random number).

Value

A string with the name of a variable.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

systemDir, systemFile

Examples

```
tempvar()
```

toRjson *Convert R object to and from RJSON specification*

Description

RJSON is an object specification that is not unlike JSON, but better adapted to represent R objects (i.e., richer than JSON). It is also easier to parse and evaluate in both R and JavaScript to render the objects in both languages. RJSON objects are used by SciViews to exchange data between R and SciViews GUIs like Komodo/SciViews-K.

Usage

```
toRjson(x, attributes = FALSE)
evalRjson(rjson)
listToJson(x)
```

Arguments

x	any R object to be converted into RJSON (do not work with objects containing C pointers, environments, promises or expressions, but should work with almost all other R objects).
attributes	if FALSE (by default), a simple object is created by ignoring all attributes. This is usally the suitable option to transfer data to another language, like JavaScript that do not understand R attributes anyway. With attributes = TRUE, the complete information about the object is written, so that the object could be recreated (almost) identical when evaluated in R (but prefer save and load to tranfer objects between R sessions!).
rjson	a string containing an object specified in RJSON notation. The specification is evaluated in R... and it can contain also R code. There is no protection provided against execution of bad code. So, you must trust the source!

Details

JSON (JavaScript Object Notation) allows to specify fairly complex objects that can be rather easily exchanged between languages. The notation is also human-readable and not too difficult to edit manually (although not advised, of course). However, JSON has too many limitations to represent R objects (no NA versus NaN, no infinite numbers, no distinction between lists and objects with attributes, or S4 objects, etc.). Moreover, JSON is not very easy to interpret in R and the existing implementations can convert only specified objects (simple objects, lists, data frames, ...).

RJSON slighly modifies and enhances JSON to make it: (1) more complete to represent almost any R object (except objects with pointers, environments, ..., of course), and (2) to make it very easy to parse and evaluate in both R and JavaScript (and probably many other) languages.

With attributes = FALSE, factors and Dates are converted to their usual character representation before encoding the RJSON object. If attributes = TRUE, they are left as numbers and their attributes (class, -and levels for factor-) completely characterize them (i.e., using evalRjson() and such objects recreate factors or Dates, respectively). However, they are probably less easy to handle in JavaScript of other language where you import the RJSON representation.

Note also that a series of objects are not yet handled correctly. These include: complex numbers, the different date flavors other that Date, functions, expressions, environments, pointers. Do not use such items in objects that you want to convert to RJSON notation.

A last restriction for the moment: you cannot have any special characters like \n, \t, \f, \r, \b, \\", or \" in names. If you want to make your names most compatible with JavaScript, note that the dot is not allowed in syntactically valid names, but the dollar sign is allowed.

TODO: a complete specification of RJSON (somewhere... a wiki page?)

Value

For toRjson(), a character string vector with the RJSON specification of the argument.

For evalRjson(), the corresponding R object in case of a pure RJSON object specification, or the result of evaluating the code, if it contains R commands (for instance, a RJSONp -RJSON with padding- item where a RJSON object is an argument of an R function that is evaluated. In this case, the result of the evaluation is returned).

For `listToJson()`, correct (standard) JSON code is generated if `x` is a list of character strings, or lists.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

See Also

`parseText`

Examples

```
## A complex R object
Robj <- structure(list(
a = as.double(c(1:5, 6)),
LETTERS,
c = c(c1 = 4.5, c2 = 7.8, c3 = Inf, c4 = -Inf, NA, c6 = NaN),
c(TRUE, FALSE, NA),
e = factor(c("a", "b", "a")),
f = "this is a \"string\"\nwith\tspecial chars",
g = matrix(rnorm(4), ncol = 2),
`h&'$@` = data.frame(x = 1:3, y = rnorm(3),
fact = factor(c("b", "a", "b"))),
i = Sys.Date(),
    j = list(1:5, y = "another item")),
    comment = "My comment\n\"",
    anAttrib = 1:10,
    anotherAttrib = list(TRUE, y = 1:4))

## Convert to simplest RJSON, without attributes
Rjson1 <- toRjson(Robj)
Rjson1
evalRjson(Rjson1)

## More complex RJSON, with attributes
Rjson2 <- toRjson(Robj, TRUE)
Rjson2
Robj2 <- evalRjson(Rjson2)
Robj2
## Numbers near equivalence comparison (note: identical(Robj, Robj2) is FALSE)
all.equal(Robj, Robj2)

rm(Robj, Robj2, Rjson1, Rjson2)
```

`unitTests.svMisc` *Unit tests for the package svMisc*

Description

Performs unit tests defined in this package by running `example(unitTests.svMisc)`. Tests are in `runit*.R` files located in the '/unitTests' subdirectory or one of its subdirectories ('/inst/unitTests' and subdirectories in package sources).

Author(s)

Philippe Grosjean (<phgrosjean@sciviews.org>)

Examples

```
if (require(svUnit)) {
    clearLog()
    runTest(svSuite("package:svMisc"), "svMisc")
    errorLog()
}
```

Index