Package 'tcltk2'

October 14, 2022

Type Package

Version 1.2-11

Date 2014-12-19

Title Tcl/Tk Additions

Author Philippe Grosjean [aut, cre]

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

Depends R (>= 2.8.0), tcltk

Suggests utils

SystemRequirements Tcl/Tk (>= 8.5), Tktable (>= 2.9, optional)

Description A series of additional Tcl commands and Tk widgets with style and various functions (under Windows: DDE exchange, access to the registry and icon manipulation) to supplement the tcltk package

License LGPL-3 + file LICENSE

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

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

NeedsCompilation no

Repository CRAN

Date/Publication 2014-12-20 10:35:06

R topics documented:

`tcltk2-package` *Tcl/Tk Additions*

Description

Additions to the tcltk package: more Tk widgets and Tcl/Tk commands.

Details

Package:	tcltk2
Type:	Package
Version:	1.2-11
Date:	2014-12-19
License:	LGPL-3 plus see LICENSE file
LazyLoad:	yes

A series of additional Tk widgets with style and various functions to supplement the tcltk package.

Author(s)

Philippe Grosjean

Maintainer: Philippe Grosjean `<phgrosjean@sciviews.org>`

See Also

`tk2widgets`, `tclFun`

`setLanguage` *Change or get the language used in R and Tcl/Tk*

Description

The function changes dynamically the language used by both R (messages only) and Tcl/Tk, or it retrieves its current value.

Usage

```
setLanguage(lang)
getLanguage()
```

Arguments

`lang`	an identification for the targeted language, for instance, \"en\" for English, \"fr\" for French, \"de\" for German, \"it\" for Italian, etc.

Value

`setLanguage()` returns `TRUE` if language was successfully changed in Tcl/Tk, `FALSE` otherwise. `getLanguage()` returns a string with current language in use for R, or an empty string if it cannot determinate which is the language currently used.

Note

You need the msgcat Tcl package to use this (but it is provided with all recent distributions of Tcl/Tk by default)

Author(s)

Philippe Grosjean

Examples

```
## Determine which language is currently in use in R
oldlang <- getLanguage()
if (oldlang != "") {
    ## Switch to English and test a command that issues a warning
    if (setLanguage("en_US")) 1:3 + 1:2
    ## Switch to French and test a command that issues a warning
    if (setLanguage("fr_FR")) 1:3 + 1:2
    ## Switch to German and test a command that issues a warning
    if (setLanguage("de_DE")) 1:3 + 1:2
    ## Switch to Italian and test a command that issues a warning
    if (setLanguage("it_IT")) 1:3 + 1:2
    ## Etc..

    ## Restore previous language
    setLanguage(oldlang)
}
```

`tclTask` *Schedule and manage delayed tasks*

Description

Tcl allows fo scheduling execution of code on the next event loop or after a given time (`after` Tcl command). `tclTaskXxx()` functions use it to schedule execution of R code with much control from within R (central management of scheduled tasks, possibility to define redoable tasks, use of S3 objects to keep track of tasks information. The `tclAfterXxx()` functions are low-level access to the Tcl `after` command.

Usage

```
## Convenient tclTask objects management
tclTaskSchedule(wait, expr, id = "task#", redo = FALSE)
tclTaskRun(id)
tclTaskGet(id = NULL, all = FALSE)
tclTaskChange(id, expr, wait, redo)
tclTaskDelete(id)

## Low-level Tcl functions
tclAfter(wait, fun)
tclAfterCancel(task)
tclAfterInfo(task = NULL)
```

Arguments

`wait`	time in ms to delay the task (take care: approximative value, depends on when event loops are triggered). Using a value lower or equal to zero, the task is scheduled on the next event loop.
`fun`	name of the R function to run (you may not supply arguments to this function, otherwise it is not scheduled properly; take care of scoping, since a copy of the function will be run from within Tcl).
`expr`	an expression to run after 'wait'.
`id`	the R identifier of the task to schedule, if this id contains #, then, it is replaced by next available number, but you cannot schedule more than a thousand tasks with the same name (the system will give up well before, anyway). If `NULL` in `tclTaskGet()`, retrieve the list of all existing tasks.
`all`	if `id = NULL`, `all = TRUE` indicate to list all tasks, including hidden ones (with id starting with a dot).
`redo`	should the task be rescheduled n times, indefinitely (`redo = TRUE`) or not (`redo = FALSE`, default, or a value <= 0).
`task`	a Tcl task timer, or its name in Tcl (in the form of 'after#xxx').

Value

The `tclAfterXxx()` functions return a 'tclObj' with the result of the corresponding Tcl function. `tclAfter()` returns the created Tcl timer in this object. If 'task' does not ecxists, `tclAfterInfo()` returns `NULL`.

`tclTaskGet()` returns a 'tclTask' object, a list of such objects, or `NULL` if not found.

The four remaining `tclTaskXxx()` functions return invisibly `TRUE` if the process is done successfully, `FALSE` otherwise. `tclTaskRun()` forces running a task now, even if it is scheduled later.

Author(s)

Philippe Grosjean

See Also

`tclFun`, `addTaskCallback`, `Sys.sleep`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

## Run just once, after 1 sec
test <- function () cat("==== Hello from Tcl! ====\n")
tclTaskSchedule(1000, test())
Sys.sleep(2)

## Run ten times a task with a specified id
test2 <- function () cat("==== Hello again from Tcl! ====\n")
tclTaskSchedule(1000, test2(), id = "test2", redo = 10)
Sys.sleep(1)

## Run a function with arguments (will be evaluated in global environment)
test3 <- function (txt) cat(txt, "\n")
msg <- "==== First message ===="
tclTaskSchedule(1000, test3(msg), id = "test3", redo = TRUE)
Sys.sleep(2)
msg <- "==== Second message ===="
Sys.sleep(2)

## Get info on pending tasks
tclTaskGet() # List all (non hidden) tasks
tclTaskGet("test2")
## List all active Tcl timers
tclAfterInfo()

## Change a task (run 'test3' only once more, after 60 sec)
tclTaskChange("test3", wait = 60000, redo = 1)
Sys.sleep(1)
## ... but don't wait so long and force running 'test3' right now
tclTaskRun("test3")
```

```
Sys.sleep(3)
## finally, delete all pending tasks
tclTaskDelete(NULL)

## End(Not run)
```

`tclVarFun` *Manipulate R variables and functions from tcl and back*

Description

These functions are intended to provide a better "duality" between the name of variables in both R and tcl, including for function calls. It is possible to define a variable with the same name in R and tcl (the content is identical, but copied and coerced in the two respective environments). It is also possible to get the value of a tcl variable from R, and to call a R function from within tcl. These functionnalities are provided in the tcltk package, but Tcl variable usually have different internal names as R equivalents.

Usage

```
makeTclNames(names, unique = FALSE)
tclFun(f, name = deparse(substitute(f)))
tclGetValue(name)
tclSetValue(name, value)
tclVarExists(name)
tclVarFind(pattern)
tclVarName(name, init = "", keep.existing = TRUE)
```

Arguments

`names`	transform names so that they are valid for variables in Tcl.
`unique`	should these names be unique in the vector?
`f`	an R function. currently, do no support functions with arguments.
`name`	the name of a variable.
`value`	The value to place in a variable.
`pattern`	a pattern to search for.
`init`	initial value to use when creating the variable.
`keep.existing`	if the tcl variable already exist, should we keep its content?

Details

These functions are similar to `tclVar()` from package tcltk, except for the following change: here, it is possible to propose a name for the created tcl variable, or to set or retrieve the content of a tcl variable that is not mirrored in R.

Value

Most of these functions return a `tclVar` object.

Author(s)

Philippe Grosjean

See Also

`tk2edit`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

## Tcl functions and variables manipulation
tclVarExists("tcl_version")
tclVarExists("probably_non_existant")
tclVarFind("tcl*")

## Using tclVarName() and tclGetValue()...
## intented for better match between R and Tcl variables
Test <- tclVarName("Test", "this is a test!")
## Now 'Test' exist both in R and in Tcl... In R, you need to use
tclvalue(Test) # to retrieve its content
## If a variable already exists in Tcl, its content is preserved using
## keep.existing = TRUE

## Create a variable in Tcl and assign "just a test..." to it
tclSetValue("A_Variable", "just to test...")
## Create the dual variable with same name
A_Variable <- tclVarName("A_Variable", "something else?")
tclvalue(A_Variable) # Content of the variable is not changed!

## If you want to retrieve the content of a Tcl variable,
## but do not want to create a reference to it in R, use:

## Create a Tcl variable, not visible from R
tclSetValue("Another_Variable", 1:5)
tclGetValue("Another_Variable") # Get its content in R (no conversion!)
tclSetValue("Another_Variable", paste("Am I", c("happy", "sad"), "?"))
tclGetValue("Another_Variable") # Get its content in R (no conversion!)

## End(Not run)
```

tk2commands *Tk commands associated with the tk2XXX widgets*

Description

These commands supplement those available in the tcltk package to ease manipulation of tk2XXX widgets.

Usage

```
tk2column(widget, action = c("add", "configure", "delete", "names", "cget",
    "nearest"), ...)
tk2insert.multi(widget, where = "end", items)
tk2list.delete(widget, first, last = first)
tk2list.get(widget, first = 0, last = "end")
tk2list.insert(widget, index = "end", ...)
tk2list.set(widget, items)
tk2list.size(widget)
tk2notetraverse(nb)
tk2notetab(nb, tab)
tk2notetab.select(nb, tab)
tk2notetab.text(nb)
tk2state.set(widget, state = c("normal", "disabled", "readonly"))
is.tk()
is.ttk()
tk2theme.elements()
tk2theme.list()
tk2theme(theme = NULL)
tk2style(class, style, state = c("default", "active", "disabled", "focus",
    "!focus", "pressed", "selected", "background", "readonly", "alternate",
    "invalid", "hover", "all"), default = NULL)
tk2dataList(x)
tk2configList(x)
```

Arguments

`widget`	the widget to which these actions apply.
`action`	which kind of action?
`where`	where are these item added in the list (by default, at the end).
`items`	the items to add (either a vector for a single line, or a matrix for more items).
`...`	further arguments to the action.
`first`	the 0-based first index to consider in the list.
`last`	the 0-based last index to consider in the list, or `"end"` for using the last element of the list.
`index`	the 0-based index where to insert items in the list.

`nb`	a tk2notebook or ttk2notebook widget (\"tclObj\" object).
`tab`	the name (text) of a tab in a notebook.
`state`	the new state of the widget, or the state to inquiry.
`theme`	a theme to use (character string).
`class`	the class of the tk2widget (either the Tk class, like `TButton`, or the name of the function that creates it, like `tk2button`)
`style`	a character string with the name of the style to retrieve
`default`	the default value to return in case this style is not found
`x`	either a tk2widget object, or a character string with its class name.

Details

`tk2column()` manipulate columns of a tk2mclistbox widget, `tk2insert.multi()` is used to insert multiple field entries in a tk2mclistbox widget, `is.tk()` determines if the tk package is loaded (on some platforms it is possible to load the tcltk package without tk, for instance, in batch mode). `is.ttk()` determines if 'ttk' widgets (styled widgets) used by the `tk2XXX()` functions are available (you need Tk >= 8.5).

Note

In comparison with traditional Tk widgets, ttk proposes an advances mechanism for styling the widgets with \"themes\". By default, it adapts to the current platform (for instance, under Windows XP with XP theme, all widgets take the appearance of XP themed widgets (even with custom themes applied!). Usual Tk widgets are ALWAYS displayed in old-looking fashion under Windows XP. If you want, you can switch dynamically to a different theme among those avaiable (list them using `tk2theme.list()`, and switch to another one with `tk2theme(newtheme)`. This is most useful to see how your GUI elements and dialog boxes look like on foreign systems. If you prefer, let's say, a Unix look of the R GUI elements under Windows, these functions are also useful. If you are more advanturous, you can even design your own themes (see the tile documentation on the Tcl wiki).

Author(s)

Philippe Grosjean

See Also

`tk2widgets`, `tk2tip`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

tt <- tktoplevel()
# A label with a image and some text
file <- system.file("gui", "SciViews.gif", package = "tcltk2")
```

```
# Make this a tk2image function...
Image <- tclVar()
tkimage.create("photo", Image, file = file)

tlabel <- tk2label(tt, image = Image,
text = "A label with an image")
tkpack(tlabel)
config(tlabel, compound = "left")

tlabel2 <- tk2label(tt, text = "A disabled label")
tkpack(tlabel2)
disabled(tlabel2) <- TRUE

fruits <- c("Apple", "Orange", "Banana")
tcombo <- tk2combobox(tt, values = fruits)
tkpack(tcombo)
tkinsert(tcombo, 0, "Apple")

## Buttons
tbut <- tk2button(tt, text = "Enabled")
tbut2 <- tk2button(tt, text = "Disabled")
tkpack(tbut, tbut2)
tkconfigure(tbut2, state = "disabled")

tcheck <- tk2checkbutton(tt, text = "Some checkbox")
tcheck2 <- tk2checkbutton(tt, text = "Disabled checkbox")
tkconfigure(tcheck2, state = "disabled")
tcheck3 <- tk2checkbutton(tt, text = "Disabled and selected")
tkpack(tcheck, tcheck2, tcheck3)
cbValue <- tclVar("1")
tkconfigure(tcheck3, variable=cbValue)
tkconfigure(tcheck3, state = "disabled")

tradio <- tk2radiobutton(tt, text = "Some radiobutton")
tradio2 <- tk2radiobutton(tt, text = "Disabled and checked")
tkpack(tradio, tradio2)
tkconfigure(tradio2, state = "checked")
tkconfigure(tradio2, state = "disabled")

## Menu allowing to change ttk theme
topMenu <- tkmenu(tt)           # Create a menu
tkconfigure(tt, menu = topMenu) # Add it to the 'tt' window
themes <- tk2theme.list()
themeMenu <- tkmenu(topMenu, tearoff = FALSE)
if ("alt" %in% themes) tkadd(themeMenu, "command", label = "alt",
    command = function() tk2theme("alt"))
if ("aqua" %in% themes) tkadd(themeMenu, "command", label = "aqua",
    command = function() tk2theme("aqua"))
if ("clam" %in% themes) tkadd(themeMenu, "command", label = "clam",
    command = function() tk2theme("clam"))
tkadd(themeMenu, "command", label = "clearlooks",
    command = function() tk2theme("clearlooks"))
if ("classic" %in% themes) tkadd(themeMenu, "command", label = "classic",
    command = function() tk2theme("classic"))
if ("default" %in% themes) tkadd(themeMenu, "command", label = "default",
    command = function() tk2theme("default"))
tkadd(themeMenu, "command", label = "keramik",
    command = function() tk2theme("keramik"))
tkadd(themeMenu, "command", label = "plastik",
    command = function() tk2theme("plastik"))
tkadd(themeMenu, "command", label = "radiance (fonts change too)!",
    command = function() tk2theme("radiance"))
if ("vista" %in% themes) tkadd(themeMenu, "command", label = "vista",
    command = function() tk2theme("vista"))
if ("winnative" %in% themes) tkadd(themeMenu, "command", label = "winnative",
    command = function() tk2theme("winnative"))
if ("xpnative" %in% themes) tkadd(themeMenu, "command", label = "xpnative",
    command = function() tk2theme("xpnative"))
tkadd(themeMenu, "separator")
tkadd(themeMenu, "command", label = "Quit", command = function() tkdestroy(tt))
tkadd(topMenu, "cascade", label = "Theme", menu = themeMenu)
tkfocus(tt)

## End(Not run)
```

tk2dde *Use DDE (Dynamic Data Exchange) under Windows*

Description

These functions are Windows-specific. They issue an error under a different platform.

Usage

```
tk2dde(topic = NULL)
tk2dde.services(service = "", topic = "")
tk2dde.exec(service, topic, command, async = FALSE)
tk2dde.poke(service, topic, item, data)
tk2dde.request(service, topic, item, binary = FALSE)
```

Arguments

`topic`	the 'topic' to reach or expose. A DDE server is accessed as 'service'l'topic'. In the case of `tk2DDE()`, a non null topic activates the DDE server, and a null topic deactivate it.
`service`	the name of the service to reach. In `tk2DDEservices`, if both service and topic are empty, the list of all available DDE service is returned, otherwise, only available topics for a given service are listed.
`command`	a string with the command to run in the external application (syntax depends on the server).

`async`	is a command run asynchroneously (returns immediately, before the command is processed), or not?
`item`	the concerned item (usually a variable name, a range in a worksheet, etc...).
`data`	the new value for the item.
`binary`	should the return bez treated as binary data or not?

`tk2dialogs` *Additional Tk dialog boxes*

Description

Tk dialog boxes to select a font or a list of ordered items.

Usage

```
tk2chooseFont(...)
tk2swaplist(items, selection, title = "Select items", ...)
```

Arguments

`items`	a vector of numbers, logicals, characters, factor or ordered.
`selection`	preselected items, in the right order.
`title`	title of the dialog box.
`...`	further arguments passed to the dialog box.

Value

The selection made in the dialog box if `OK` is clicked, `""` otherwise for `tk2chooseFont()`, or an zero-length vector for `tk2swaplist()`.

Note

If you use tile 0.7.2 or above, these dialog boxes will automatically use it. Otherwise, the dialog boxes will use plain Tk widgets (not yet for `tk2swaplist()`).

Author(s)

Philippe Grosjean

See Also

`tk2widgets`, `tk2commands`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

tk2chooseFont()
tk2chooseFont(font = "{courier} 9", title = "Choose a fixed font",
    fonttype = "fixed", style = 4, sizetype = "all")
tk2chooseFont(font = "Verdana 12 bold italic underline overstrike",
    fonttype = "prop", style = 2, sizetype = "point")

tk2swaplist(1:10, 1:5) # integer
tk2swaplist(as.numeric(1:10), 1:5) # double
tk2swaplist(paste("V", 1:10), paste("V", 1:5)) # character
tk2swaplist(as.factor(1:10), 1:5) # factor
tk2swaplist(as.ordered(1:10), 1:5) # ordered

## End(Not run)
```

`tk2edit` *Edit a matrix or data frame in spreadsheet-like editor*

Description

A tkTable widget is used to display and edit a matrix or data frame. One can edit entries, add or delete rows and columns, ...

Usage

```
tk2edit(x, title = "Matrix Editor", header = NULL,
    maxHeight = 600, maxWidth = 800, fontsize = 9, ...)
```

Arguments

`x`	a matrix or data frame to edit.
`title`	the title of the editor window.
`header`	do we display a header?
`maxHeight`	the maximum height of the editor window.
`maxWidth`	the maximum width of the editor window.
`fontsize`	the size of the font to use in the editor window.
`...`	further arguments to pass to the function.

Value

The function is used for its side-effet, that is, to modify a matrix or data frame in a spreadsheet-like editor.

Note

You need the tkTable widget to use this function

Author(s)

Jeffrey J. Hallman

See Also

`tclSetValue`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded
data(iris)
tk2edit(iris)

## End(Not run)
```

`tk2fonts` *Edit a matrix or data frame in spreadsheet-like editor*

Description

A tkTable widget is used to display and edit a matrix or data frame. One can edit entries, add or delete rows and columns, ...

Usage

```
tk2font.get(font, what = c("family", "size", "bold", "italic"))
tk2font.set(font, settings)
tk2font.setstyle(text = TRUE, system = FALSE, default.styles = FALSE)
```

Arguments

`font`	the name of one or several cached Tk font.
`what`	a list of font characteristics to get: 'family', 'size', 'bold', 'italic', 'underline' and/or 'overstrike'. By default, everything except 'underline' and 'overstrike'.
`settings`	settings of fonts. There are two forms possibles: (1) a vector of character strings of same length as font with Tk fonts description like '-family Times -size 12 -weight bold', for instance, or (2) a list of font characteristics (list with components 'family', 'size', 'bold', 'italic', 'underline' and 'overstrike').
`text`	do we synchronise text Tk fonts (text, titles, and fixed-font text) with current settings, as in `.Fonts` in `SciViews:TempEnv`?

`system`	do we synchronise system Tk fonts (widgets, window caption, menus, tooltips, ...) with current system configuration? This is highly platform dependent. Currently, system settings are gathered only under Windows, thanks to the winSystemFonts() function.
`default.styles`	do we add `.fontsStyleXXX` in `SciViews:TempEnv`, where XXX is one of the four default styles: 'Classic', 'Alternate', 'Presentation' or 'Fancy'.

Value

`tk2font.get()` retrieves a list with font characteristics (same format as the `settings` argument) for the first Tk font found in its `font` argument, or `""` if the font is not found. `tk2font.set()` changes current font settings or, possibly, create the Tk font. `tk2font.setstyle()` changes the current Tk fonts settings according to actual system and/or text configuration fonts.

Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded
## Refresh both text and system Tk fonts
tk2font.setstyle(system = TRUE)
## Get characteristics of the default font
tk2font.get("TkDefaultFont")

## End(Not run)
```

`tk2ico` *Manipulate icons under Windows*

Description

These function are only useful for Windows, but they can be used without error on other platform for making platform-independent code that has an additional behaviour under Windows. On the other platforms, these function just return NULL silently.

Usage

```
tk2ico.create(iconfile, res = 0, size = 16)
tk2ico.destroy(icon)
tk2ico.list(file = "shell32.dll")
tk2ico.sizes(file = "shell32.dll", res = "application")
tk2ico.load(file = "shell32.dll", res = "application", size = 16)
tk2ico.set(win, icon, pos = NULL, type = c("all", "small", "big"))
tk2ico.setFromFile(win, iconfile)
```

```
## Deprecated functions since drop of winico.dll support
tk2ico.hicon(icon)
tk2ico.info(icon, convert = TRUE)
tk2ico.text(icon)
tk2ico.text(icon) <- value
tk2ico.pos(icon) <- value
tk2ico.taskbar.add(icon, pos = 0, text = tk2ico.text(icon),
    leftmenu = NULL, rightmenu = NULL)
tk2ico.taskbar.delete(icon)
tk2ico.taskbar.modify(icon, pos = NULL, text = NULL)
```

Arguments

`iconfile`	a file with a .ico, or .exe extension, containing one or more Windows icons
`file`	a file having icon resources (.exe, or .dll)
`res`	the name of the resource from where the icon should be extracted
`size`	the size of the icon to use. For windows icons, 16 should be fine usually
`win`	a Tk window, or an integer representing the handle (HWND) of a foreign window whose icon will be changed (take care, the function returns `TRUE` even if the handle is wrong!
`icon`	an icon object.
`convert`	do we convert the result into a data.frame?
`pos`	a position (starting from 0) pointing to an icon in a multi-icon object. Note that `pos` is not used in `tk2ico.set()` if `type = "all"` (in this case, best icons matching both \"small\" and \"large\" sizes are searched in the `icon` resource.
`type`	do we change only the small, the large, or both icons?
`value`	a string with the new text for the icon in `tk2ico.text()` or a numerical value indicating the new default position in the icon resource for `tk2ico.pos()`.
`text`	change a text for an icon.
`leftmenu`	a \"tkwin\" object to display when the user left-clicks on the taskbar icon (usually, a Tk popup menu), or `NULL` for nothing.
`rightmenu`	idem as \’lefmenu\’ but for a right-click on the taskbar icon.

Value

These function do nothing and return `NULL` on other platforms than Windows.

Author(s)

Philippe Grosjean

`tk2methods` *A series of methods applicable to tk2widget or tk2cfglist objects*

Description

Tk2widgets can be used as tcltk widgets, but they propose also an object-oriented interaction through these different methods.

Usage

```
is.tk2widget(x)
## S3 method for class 'tk2widget'
print(x, ...)

tk2cfglist(...)
## S3 method for class 'tk2cfglist'
print(x, ...)

state(x, ...)
## S3 method for class 'tk2widget'
state(x, ...)

label(x, ...)
## S3 method for class 'tk2widget'
label(x, ...)
label(x) <- value
## S3 replacement method for class 'tk2widget'
label(x) <- value

tag(x, ...)
## S3 method for class 'tk2widget'
tag(x, ...)
tag(x) <- value
## S3 replacement method for class 'tk2widget'
tag(x) <- value

disabled(x, ...)
## S3 method for class 'tk2widget'
disabled(x, ...)
disabled(x) <- value
## S3 replacement method for class 'tk2widget'
disabled(x) <- value

values(x, ...)
## S3 method for class 'tk2widget'
values(x, ...)
## S3 method for class 'tk2listbox'
```

```
values(x, ...)
values(x) <- value
## S3 replacement method for class 'tk2widget'
values(x) <- value
## S3 replacement method for class 'tk2listbox'
values(x) <- value

value(x, ...)
## S3 method for class 'tk2widget'
value(x, ...)
## S3 method for class 'tk2listbox'
value(x, ...)
value(x) <- value
## S3 replacement method for class 'tk2widget'
value(x) <- value
## S3 replacement method for class 'tk2listbox'
value(x) <- value

selection(x, ...)
## S3 method for class 'tk2widget'
selection(x, ...)
## S3 method for class 'tk2listbox'
selection(x, ...)
selection(x) <- value
## S3 replacement method for class 'tk2widget'
selection(x) <- value
## S3 replacement method for class 'tk2listbox'
selection(x) <- value

visibleItem(x, index, ...)
## S3 method for class 'tk2widget'
visibleItem(x, index, ...)
## S3 method for class 'tk2listbox'
visibleItem(x, index, ...)

size(x, ...)
## S3 method for class 'tk2widget'
size(x, ...)
## S3 method for class 'tk2listbox'
size(x, ...)

config(x, ...)
## S3 method for class 'tk2widget'
config(x, cfglist, ...)
## S3 method for class 'tk2label'
config(x, cfglist, ...)
config(x) <- value
## S3 replacement method for class 'tk2widget'
```

```
config(x) <- value
## S3 replacement method for class 'tk2label'
config(x) <- value
```

Arguments

`x`	a tk2widget object.
`...`	a series of named arguments corresponding to parameters and values to use for the configuration for `tk2cfglist()`, or reserved arguments for future use for the other function (not used yet).
`value`	a value to assign to the object's method.
`index`	the index of the item in the list to make visible.
`cfglist`	a named list with configuration parameters and values to apply (usually, a `tk2cfglist` object).

Value

Depnds on the function (TODO: complete this...).

Author(s)

Philippe Grosjean

See Also

`tk2widgets`, `tk2tip`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

## TODO: examples illustrating the use of these methods...

## End(Not run)
```

`tk2reg` *Manipulate the registry under Windows*

Description

These functions are Windows-specific. They issue an error on other platforms.

Usage

```
tk2reg.broadcast()
tk2reg.delete(keyname, valuename)
tk2reg.deletekey(keyname)
tk2reg.get(keyname, valuename)
tk2reg.keys(keyname)
tk2reg.set(keyname, valuename, data, type = c("sz", "expand_sz", "multi_sz",
    "dword", "dword_big_endian"))
tk2reg.setkey(keyname)
tk2reg.type(keyname, valuename)
tk2reg.values(keyname)
```

Arguments

`keyname`	the name of the key.
`valuename`	a value in this key.
`data`	the data to place in the value.
`type`	the type of value in the registry. By default, it is 'sz', that is, an atomic string.

---

tk2tip *Display and manage tooltips in Tk widgets*

Description

`tk2tip` provides a simple mechanism to display tooltips on Tk widgets when the mouses hoover on top of them.

Usage

```
tk2tip(widget, message)
tk2killtip()
tip(x, ...)
## S3 method for class 'tk2widget'
tip(x, ...)
tip(x) <- value
## S3 replacement method for class 'tk2widget'
tip(x) <- value
```

Arguments

`widget`	the widget to which a tooltip is attached.
`message`	the message of the tooltip ("" to remove the tooltip).
`x`	a tk2widget object.
`...`	further arguments to the method (unused, but reserved for future use).
`value`	the message of the tooltip, or "" to remove the tip.

Note

This implementation is done in pure Tcl code

Author(s)

Philippe Grosjean

See Also

tk2widgets, tk2methods

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

## Using plain Tcl/Tk label and button (tk2XXX equivalent have built-in
## tooltip features)
tt <- tktoplevel()
lb <- tklabel(tt, text = "Move mouse over me, or over the button to see tooltip")
tkgrid(lb)
tk2tip(lb, "A tooltip for the label \ndisplayed on two lines")
but <- tkbutton(tt, text = "Exit", width = 10,
    command = function() tkdestroy(tt2))
tkgrid(but)
tk2tip(but, "Exit from this dialog box")

## To test tk2killtip(), move mouse on top of a widget
## so that the tip is visible, and force killing it manually using:
tk2killtip()
## Move again to the widget: the tip is displayed again.

## With tk2widgets, the tip() method can also be used:
lb2 <- tk2label(tt, text = "Move also over me to see the tooltip")
tkgrid(lb2)
tip(lb2) # No tip yet
tip(lb2) <- "Now the tooltip is there!"
## Move the mouse over that last label

tip(lb2) # Yes, this is my tooltip
tip(lb2) <- NULL # To eliminate the toltip for this widget

## End(Not run)
```

`tk2widgets` *A series of versatile using either themable ttk widgets*

Description

A series of widgets you can use in your Tk windows/dialog boxes.

Usage

```
tk2button(parent, tip = "", ...)
tk2canvas(parent, tip = "", ...)
tk2checkbutton(parent, tip = "", ...)
tk2combobox(parent, tip = "", ...)
tk2entry(parent, tip = "", ...)
tk2frame(parent, ...)
tk2label(parent, tip, label, tag, cfglist, wrap = FALSE, ...)
tk2labelframe(parent, ...)
tk2listbox(parent, values, value, selection, selectmode = c("extended", "single",
    "browse", "multiple"), height = 5, tip = "", scroll = "both",
    autoscroll = "x", enabled = TRUE, ...)
tk2mclistbox(parent, tip ="", ...)
tk2menu(parent, activebackground, activeforeground, ...)
tk2menubutton(parent, tip = "", ...)
tk2message(parent, text = "", justify = c("left", "center", "right"),
    width = -1, aspect = 150, tip = "", ...)
tk2notebook(parent, tabs, ...)
tk2panedwindow(parent, orientation = c("horizontal", "vertical"), ...)
tk2progress(parent, orientation = c("horizontal", "vertical"), tip = "", ...)
tk2radiobutton(parent, tip = "", ...)
tk2scale(parent, orientation = c("horizontal", "vertical"), tip = "", ...)
tk2scrollbar(parent, orientation = c("horizontal", "vertical"), ...)
tk2separator(parent, orientation = c("horizontal", "vertical"), ...)
tk2spinbox(parent, tip = "", ...)
tk2table(parent, ...)
tk2tablelist(parent, ...)
tk2text(parent, tip = "", ...)
tk2ctext(parent, tip = "", ...)
tk2tree(parent, tip = "", ...)
```

Arguments

`parent`	the parent window.
`tip`	a tooltip to display for this widget (optional).
`label`	a single character string used to label that widget (optional).
`tag`	any object that you would like to associate with this widget (optional).
`cfglist`	a named list with configuration parameters and values to apply.
`wrap`	do we wrap long lines in the widget?
`values`	a character vector with values to use to populate the widget.
`value`	a character vector with current value for the widget, or currently selected values, if multiple selection is allowed. Takes precedence on `selection`.

`selection`	a numeric (indices) vector with current selection.
`selectmode`	the selection mode for this widget. `extended` is the usual choice for multi-selction `tk2listbox()`.
`height`	the height of the widget.
`scroll`	do we add scrollbars? Possible values are `"x"`, `"y"`, `"both"` or `"none"`; can be abridged.
`autoscroll`	do we automatically hide scrollbars if not needed? Possible values are the same as for the `scroll` argument.
`enabled`	is the widget enabled or disabled?
`text`	the text to display in the widget.
`justify`	how text is justified?
`tabs`	the tabs to create in the notebook widget.
`width`	the desired width. Use a negative value to use `aspect` instead.
`aspect`	sets the aspect ratio of the widget (100 = square, 200 = twice as large, 50 = twice as tall). Only used if `width` is negative.
`orientation`	either `"horizontal"` or `"vertical"`.
`activebackground`	color to use for active background of menu items (if not provided, a reasonable default value is used).
`activeforeground`	color to use for active foreground of menu items (if not provided, a reasonable default value is used).
`...`	further arguments passed to the widget.

Value

The reference to the created widget.

Note

You need Tk 8.5 or above to use these widgets.

Author(s)

Philippe Grosjean

See Also

`is.ttk`

Examples

```
## Not run:
## These cannot be run by examples() but should be OK when pasted
## into an interactive R session with the tcltk package loaded

### A tk2notebook example
tt2 <- tktoplevel()
nb <- tk2notebook(tt2, tabs = c("Test", "Button"))
tkpack(nb, fill = "both", expand = 1)
tb1 <- tk2notetab(nb, "Test")
lab <- tk2label(tb1, text = "Nothing here.")
tkpack(lab)
tb2 <- tk2notetab(nb, "Button")
but <- tk2button(tb2, text = "Click me", command = function() tkdestroy(tt2))
tkgrid(but)
tk2notetab.select(nb, "Button")
tk2notetab.text(nb) # Text of the currently selected tab

### A simple tk2panedwindow example
tt2 <- tktoplevel()
pw <- tk2panedwindow(tt2, orient = "vertical")
lpw.1 <- tk2text(pw)
lpw.2 <- tk2text(pw)
tkadd(pw, lpw.1)#, minsize = 100)
tkadd(pw, lpw.2)#, minsize = 70)
but <- tk2button(tt2, text = "OK", width = 10,
    command = function() tkdestroy(tt2))
tkpack(pw, fill = "both", expand = "yes")
tkpack(but)
## Resize the window and move the panel separator with the mouse

### A tk2combobox example
tt2 <- tktoplevel()
cb <- tk2combobox(tt2)
tkgrid(cb)
## Fill the combobox list
fruits <- c("Apple", "Orange", "Banana")
tk2list.set(cb, fruits)
tk2list.insert(cb, "end", "Scoubidou", "Pear")
tk2list.delete(cb, 3)   # 0-based index!
tk2list.size(cb)
tk2list.get(cb)   # All items
## Link current selection to a variable
Fruit <- tclVar("Pear")
tkconfigure(cb, textvariable = Fruit)
## Create a button to get the content of the combobox
but <- tk2button(tt2, text = "OK", width = 10,
    command = function() {tkdestroy(tt2); cat(tclvalue(Fruit), "\n")})
tkgrid(but)

### An example of a tk2spinbox widget
tt2 <- tktoplevel()
```

```
tspin <- tk2spinbox(tt2, from = 2, to = 20, increment = 2)
tkgrid(tspin)
## This widget is not added yet into tcltk2!
#tdial <- tk2dial(tt2, from = 0, to = 20, resolution = 0.5, width = 70,
# tickinterval = 2)
#tkgrid(tdial)
tbut <- tk2button(tt2, text = "OK", width = 10,
    command = function() tkdestroy(tt2))
tkgrid(tbut)

### A tk2mclistbox example
tt2 <- tktoplevel()
mlb <- tk2mclistbox(tt2, width = 55, resizablecolumns = TRUE)
## Define the columns
tk2column(mlb, "add", "name", label = "First name", width = 20)
tk2column(mlb, "add", "lastname", label = "Last name", width = 20)
tk2column(mlb, "add", "org", label = "Organisation", width = 15)
tkgrid(mlb)
## Fill the multicolumn list (we can use a vector, or a matrix of character strings)
item1 <- c("Bryan", "Oackley", "ChannelPoint")
items <- matrix(c("John", "Ousterhout", "Scriptics", "Steve", "Miller", "TclTk inc."),
    ncol = 3, byrow = TRUE)
tk2insert.multi(mlb, "end", item1)
tk2insert.multi(mlb, "end", items)
#### TODO: bind events
### Ex: .listbox label bind date <ButtonPress-1> "sortByDate
### See the example.tcl in .\libs\mclistbox1.02 for a more complex example
### Create a button to close the dialog box
but <- tk2button(tt2, text = "OK", width = 10,
    command = function() tkdestroy(tt2))
tkgrid(but)

### A simple tk2table example (Tktable is required here!)
myRarray <- c("Animal", "\"sphinx moth\"", "oyster", "Type", "insect", "mollusk")
dim(myRarray) <- c(3, 2)
for (i in (0:2))
    for (j in (0:1))
        .Tcl(paste("set tclarray(", i, ",", j, ") ", myRarray[i+1, j+1], sep = ""))
tt2 <- tktoplevel()
table1 <- tk2table(tt2, variable = "tclarray", rows = "3", cols = "2",
    titlerows = "1", selectmode = "extended", colwidth = "25", background = "white")
tkpack(table1)

## A tablelist example
tt <- tktoplevel()
tlist <- tk2tablelist(tt, columntitles = c("First column", "Second column"),
    stretch = "all", expand = 1)
tkpack(tlist, fill = "both")
tkinsert(tlist, "end", c("first row", "another value"))
tkinsert(tlist, "end", c("another row", "bla bla"))
tbut <- tk2button(tt, text = "Done", command = function () tkdestroy(tt))
tkpack(tbut)
```

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

Index