Package: tcltk2 (via r-universe)

November 18, 2026

Type Package

Version 1.6.1

Title Tcl/Tk Additions

Description A series of additional Tcl commands and Tk widgets to supplement the tcltk package.

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 4.2.0), tcltk (>= 4.2.0)

Suggests utils (>= 4.2.0), covr (>= 3.5.0), knitr (>= 1.42), rmarkdown (>= 2.21), spelling (>= 2.2.1), testthat (>= 3.0.0)

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

License LGPL-3

URL https://github.com/SciViews/tcltk2/, https://www.sciviews.org/tcltk2/, https://sciviews.r-universe.dev/tcltk2

BugReports https://github.com/SciViews/tcltk2/issues/

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Encoding UTF-8

Language en-US

ByteCompile yes

Config/testthat/edition 3

Config/pak/sysreqs tcl tk tk-dev tk-table

Repository https://sciviews.r-universe.dev

Date/Publication 2025-08-29 06:07:32 UTC

RemoteUrl https://github.com/SciViews/tcltk2

RemoteRef HEAD

RemoteSha d3f569ae24981969dae96eff6f642e8071a5cac7

Contents

is.tk2widget 2
makeTclNames 5
setLanguage 7
tclAfter 9
tk2button 12
tk2chooseFont 17
tk2column 18
tk2dde 22
tk2edit 25
tk2font.get 26
tk2ico.create 28
tk2reg.broadcast 29
tk2swaplist 32
tk2tip 33

Index **36**

`is.tk2widget` *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 zero-based index of the item to make visible.
cfglist	a list containing one or more named items, with the name being a Tcl/Tk property and items being the new value for the property.

Value

Depends on the function. The `is.xxx()` function return `TRUE` or `FALSE` if the object is of the right class or not. The assignations form return the assigned value. The direct form return the item.

Author(s)

Philippe Grosjean

See Also

`tk2button()`, `tk2tip()`

`makeTclNames` *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 features 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 `tcltk::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()`, `tcltk::tclVar()`

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:

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)
```

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

Description

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

Usage

```
setLanguage(lang)

getLanguage()
```

```
tclmclocale(lang)

tclmcset(lang, msg, translation)

tclmc(fmt, ..., domain = NULL)
```

Arguments

lang	An identification for the targeted language, for instance, \"en\" for English, \"en_US\" for american English, \"fr\" for French, \"de\" for German, \"it\" for Italian, etc. Facultative argument for `tclmclocale()`.
msg	A single character string with the message to translate.
translation	The corresponding version in `lang`. Substitutions markers like `\base::gettextf()`). These translations are added in the Tcl catalog in the main domain, i.e., you don't need to give a domain name with `tclmc()` to retrieve the translation.
fmt	A single character vector of format string.
...	Values to be passed into `fmt` for the substitution.
domain	The 'domain", i;e., Tcl namespace where the translation is defined. Use `NULL` (the default) or `""` for the main domain where translations using `tclmcset()` are stored.

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, and a `tcl.language` attribute with the different catalogs that are used in priority order (ending with `""` for no translation, i.e., Tcl translations do not return an error, but the initial string if the item is not found in the catalog). `tclmclocale()` allows to change and get language for Tcl only, without changing anything for R.

The two functions `tclmcset()` and `tclmc()` allow to record and retrieve the translation of strings in the main R domain. Moreover, `tclmc()` also allows to retrieve translations of Tcl strings in other Tcl namespaces (a.k.a., domains), see the examples.

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

```
# What is the language used by Tcl?
tclmclocale()

# Define a simple translation in French and German
```

```
tclmcset("de", "Yes", "Ja")
tclmcset("fr", "Yes", "Oui")

# Determine which language is currently in use in R
(oldlang <- getLanguage())
if (oldlang != "") {
  # Switch to English; test a command that issues a warning and a Tcl string
  setLanguage("en_US")
  1:3 + 1:2
  tclmc("Yes")

  # Switch to German and test
  setLanguage("de")
  1:3 + 1:2
  tclmc("Yes")

  # Switch to Belgian French and test
  setLanguage("fr_BE")
  1:3 + 1:2
  tclmc("Yes")

  # A more complex trnaslation message with a substitution
  tclmcset("fr", "Directory contains %d files",
    "Le repertoire contient %d fichiers")
  tclmc("Directory contains %d files", 9)
  # or from a R/Tcl variable...
  nfiles <- tclVar(12)
  tclmc("Directory contains %d files", tclvalue(nfiles))

  # Retrieve a translation defined in the "tk" domain
  tclmc("Replace existing file?", domain = "tk")

  # Tcl dialog boxes are translated according to the current language
  ## Not run:
    tkgetOpenFile()

## End(Not run)

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

`tclAfter` *Schedule and manage delayed tasks*

Description

Tcl allows for 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

```
tclAfter(wait, fun)

tclAfterCancel(task)

tclAfterInfo(task = NULL)

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

tclTaskSchedule(wait, expr, id = "task#", redo = FALSE)

tclTaskRun(id)

tclTaskGet(id = NULL, all = FALSE)

tclTaskChange(id, expr, wait, redo)

tclTaskDelete(id)
```

Arguments

`wait`	Time in ms to delay the task (take care: approximate 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).
`task`	A Tcl task timer, or its name in Tcl (in the form of 'after#xxx').
`x`	A 'tclTask' object.
`...`	Further argument to the `print()` method.
`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.
`redo`	Should the task be rescheduled n times, indefinitely (`redo = TRUE`) or not (`redo = FALSE`, default, or a value <= 0).
`all`	If `id = NULL`, `all = TRUE` indicate to list all tasks, including hidden ones (with id starting with a dot).

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 exists, `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()`, `base::addTaskCallback()`, `base::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)
```

---

tk2button *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 = "", ...)

tk2text(parent, tip = "", ...)

tk2ctext(parent, tip = "", ...)

tk2tree(parent, tip = "", ...)

tk2table(parent, ...)

tk2tablelist(parent, ...)
```

Arguments

parent	The parent window.
tip	A tooltip to display for this widget (optional).
...	Further arguments passed to the widget.
label	A single character string used to label that widget (optional).
tag	An 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 multiselection `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?
`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).
`text`	The text to display in the widget.
`justify`	How text is justified?
`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.
`tabs`	The tabs to create in the notebook widget.
`orientation`	Either `"horizontal"` or `"vertical"`.

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 %W"
# 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)
```

tk2chooseFont *Additional Tk dialog boxes*

Description

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

Usage

```
tk2chooseFont(...)

tk2unicode_config(parent)

tk2unicode_select(widget)

tk2unicode_bind(widget)
```

Arguments

`...`	Further arguments passed to the dialog box.
`parent`	The Tk toplevel dialog box that will be the parent of the configuration dialog box.
`widget`	A widget that can accept a unicode character. For `tk2unicode_bind()` it must be a `tk2text` or a `tk2entry` widget.

Value

The selection made in the dialog box if `OK` is clicked, `""` otherwise for `tk2chooseFont()`.

The `tk2unicode_select()` dialog pastes the selected unicode character in the designed widget, but returns nothing. The `tk2unicode_config()` changes the configuration for the unicode composer, but returns nothing. If you decide to do so, it saves the config on a file. This is done app-by-app, and the default app name is `"R"`. You can change it by setting a different value in the option `"tk2app"`, i.e., `options(tk2app = "myApp")`. The `tk2unicode_bind()` is also invoked for its side-effect to install required bindings to enable the unicode composer engine for the given widget and it returns nothing.

Author(s)

Philippe Grosjean

See Also

`tk2text()`, `tk2listbox()`, `tk2list.insert()`

Examples

```
## Not run:
library(tcltk2)
# These cannot be run by examples() but should be OK when pasted
# into an interactive R session with the tcltk package loaded

# Font selection
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")

# Easy unicode character entry
tt <- tktoplevel()
txt <- tk2text(tt, width = 60, height = 20)
tkpack(txt)
e <- tk2entry(tt, width = 50)
tkpack(e)
# Get an unicode character for the text widget
tk2unicode_select(txt)
# and for the entry widget
tk2unicode_select(e)

# Bind the composer to both the text and the entry widgets
# and display the configuration box
# Once done, try the compose key + m + u, or compose + " + a
# or any othert sequence in both widgets
# or hit the compose key twice
tk2unicode_bind(txt)
tk2unicode_bind(e)
tk2unicode_config(tt)

## End(Not run)
```

`tk2column` *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"),
  ...
```

```
)

tk2list.set(widget, items)

tk2list.insert(widget, index = "end", ...)

tk2list.delete(widget, first, last = first)

tk2list.get(widget, first = 0, last = "end")

tk2list.size(widget)

tk2state.set(widget, state = c("normal", "disabled", "readonly"))

tk2insert.multi(widget, where = "end", items)

tk2notetraverse(nb)

tk2notetab(nb, tab)

tk2notetab.select(nb, tab)

tk2notetab.text(nb)

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)

is.tk()

is.ttk()
```

Arguments

widget	The widget to which these actions apply.

action	Which kind of action?
...	Further arguments to the action.
items	The items to add (either a vector for a single line, or a matrix for more items).
index	The 0-based index where to insert items in the list.
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.
state	The new state of the widget, or the state to inquiry.
where	Where are these item added in the list (by default, at the end).
nb	A tk2notebook widget ('tclObj' object).
tab	The name (text) of a tab in a notebook.
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).

Value

Nothing, these functions are used for their side-effect of changing the state of Tk widgets

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, all widgets take the appearance of Windows themed widgets (even with custom themes applied!). Usual Tk widgets are ALWAYS displayed in old-looking fashion under Windows. If you want, you can switch dynamically to a different theme among those available (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 adventurous, you can even design your own themes (see the tile documentation on the Tcl wiki).

Author(s)

Philippe Grosjean

See Also

tk2button(), 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

DDE is the first Microsoft's attempt to make an inter-application mechanism. It is now superseded by (D)Com, but it is still available (although declared as unsupported). Being simpler than Com, DDE is interesting for simple tasks. Applications like Word or Excel provide services one can access through DDE (see examples). This code if left for backward compatibility, and also, just in case you will find some use of it. But for new projects in general, you should not use this any more.

Usage

```
tk2dde(topic = NULL)
```

```
tk2dde.exec(service, topic, command, async = FALSE)

tk2dde.poke(service, topic, item, data)

tk2dde.request(service, topic, item, binary = FALSE)

tk2dde.services(service = "", topic = "")
```

Arguments

topic	The 'topic' to reach or expose. A DDE server is accessed as service'|'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 tk2dde.services, 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 asynchronously (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 be treated as binary data or not?

Note

This is only available under Windows. Trying to use these functions under other platforms raises an error. Under Windows, R is automatically configured as a DDE server with name 'TclEvalSciViewsR' when this package is loaded.

Author(s)

Philippe Grosjean

See Also

tk2reg.get()

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

# Examples of DDE - Windows only

### Examples using wish ###
```

```
# Start a Wish84 console side-by-side with R.
# (to get wish, you need to install ActiveTcl from
# http://www.activestate.com/Products/ActiveTcl/)
# Once it is done, start 'Wish84' from the start menu)
# Register the Wish console as a DDE server, that is, type in it
# (% is the Tcl prompt, do not type it!):
# % package require dde
# % dde servername wish

### In R:
tk2dde("R") # Return 0 if succeed
tk2dde.services()
# Evaluate some string in wish
tk2dde.exec("TclEval", "wish", "{puts {Hello World!}}")
# Give a value to a variable in wish
tk2dde.poke("TclEval", "wish", "myvar", "{This is a string!}")
# Note that you must surround strings with curly braces in Tcl!
tk2dde.poke("TclEval", "wish", "mynumvar", c(34.56, 78.9))

# In wish, check that vars exist and have correct value
# % puts $myvar
# % puts $mynumvar

# Get the value of one variable from wish into R
tk2dde.request("TclEval", "wish", "myvar")
tk2dde.request("TclEval", "wish", "mynumvar")
# Note that you do not know here if it is a string, a number, or so...
# You have to know and convert yourself!

# Now, the other way: execute a R function from wish
# You first need to register a R function for callback
# (For the moment, only functions without arguments are supported!)
doDDE <- function() cat("DDE execute!") # A simple function
tclFun(doDDE)
# And in wish
# % dde execute TclEval R doDDE

# Once you have defined a variable using tclVar, you can get or change it
# from the dde server. However, tclVar gives cryptic names like ::RTcl1.
# So we prefer to use tclVarName()
myvar2 <- tclVarName("myvar2", "this is a test...")
tclvalue(myvar2) # This is the way we access to this variable in R

# In wish you get the value and change it:
# % dde request TclEval R myvar2
# Again, dde poke does not work and must be replaced by an execute command
# This does not work (???)
# % dde poke TclEval R myvar2 {yes! and it works...}
# ... but this is fine
# % dde execute TclEval R {set myvar2 {yes! and it works...}}

# And in R...
tclvalue(myvar2)
```

```
### DDE at the command line with execdde.exe ###
# You can also change the value of a variable, or run a command in R from
# the command line using execdde.exe:
# - Download execdde.exe from http://www.sciviews.org/SciViews-R/execdde.zip
# - Unzip it and copy 'execdde.exe' somewhere in your path,
# - Start a DOS window
# - Enter the following commands ('>' is the prompt, do not type it):
# > execdde -s TclEval -t R -c doDDE > NUL
# > if errorlevel 1 echo An error occurs... branch accordingly in your batch!
# > execdde -s TclEval -t R -c "set myvar2 'ok from execdde'" > NUL

# And in R:
tclvalue(myvar2)
# Note: thanks to separate event loops, it works also when R calculates...

### Manipulating Microsoft Excel ###
# Start Excel with a blank workbook, then...

# Change values in Excel from R:
tk2dde.poke("Excel", "Sheet1", "R1C1:R2C1", c("5.7", "6.34"))   # Some data
tk2dde.poke("Excel", "Sheet1", "R3C1", "= A1 + A2")             # A formula

# Read values in Excel (note that results of formulas are returned)
Res <- tk2dde.request("Excel", "Sheet1", "R1C1:R3C1")
Res
as.numeric(Res)


## 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)
```

---

tk2font.get *Manipulate Tk fonts*

Description

Get or set fonts used by Tk widgets from within R.

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 possible forms: (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 synchronize text Tk fonts (text, titles, and fixed-font text) with current settings in .Fonts inside the SciViews:TempEnv environment?
system	Do we synchronize 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 the SciViews:TempEnv environment, 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

See Also

tk2chooseFont()

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, default.styles = TRUE)
# Get characteristics of the default font
```

```
tk2font.get("TkDefaultFont")

## End(Not run)
```

`tk2ico.create` *Manipulate icons under Windows*

Description

Create, load and work with Windows icons. Change icons for Windows. These functions are only useful for Windows, but they silently return `NULL` on other platforms for writing compatible code (Windows icons instructions can be simply ignored).

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.setFromFile(win, iconfile)

tk2ico.set(win, icon)
```

Arguments

`iconfile`	A file with a .ico, or .exe extension, containing one or more Windows icons
`res`	The name of the resource from where the icon should be extracted.
`size`	Te size of the icon to use. For windows icons, 16 should be fine usually.
`icon`	A icon object.
`file`	A file having icon resources (.exe, or .dll).
`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!

Value

An icon object, which is a reference to an image resource in Tcl. Its classes are `c("tclObj", "tclIcon")`. Do not forget to destroy it using `tk2ico.destroy()` when you do not need it any more! If `tk2ico.load()` fails, it returns `NULL` instead of a Tcl object.

Note

This is Windows-specific. It is implemented using the ico Tcl package.

Author(s)

Philippe Grosjean

See Also

tk2dde.exec(), tk2reg.get()

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

### Examples of tk2ico - icon manipulation under Windows
tt2 <- tktoplevel()
# Load a system icon (there are: "application", "asterisk", "error",
# "exclamation", "hand", "question", "information", "warning", and "winlogo".
Warn <- tk2ico.load(res = "warning")
# Change the icon of my window tt2
tk2ico.set(tt2, Warn)
# Do not forget to destroy icon to free resource when not needed any more
tk2ico.destroy(Warn)
rm(Warn)

### Otherwise, the list of icons in a file are:
tk2ico.list()
# and for a given icon, the various sizes are:
tk2ico.sizes(res = 4)

### One can set icon of a window from an .ico or .exe file directly
tk2ico.setFromFile(tt, default = file.path(R.home("bin"), "Rgui.exe"))

tk2ico.setFromFile(tt2, system.file("gui", "SciViews.ico", package = "tcltk2"))

### When done, dispose of the window and clean the workspace
tkdestroy(tt2)
rm(tt2)

## End(Not run)
```

---

tk2reg.broadcast *Manipulate the registry under Windows*

Description

These functions access the Windows registry in a secure way (most errors are handled gracefully), and ensures correct conversion back and forth for atomic strings ('sz' and 'expand_') and numbers ('dword' and 'dword_big_endian'), and for vectors of strings ('multi_sz').

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 this value.
type	The type of value in the registry. By default, it is 'sz', that is, an atomic string.

Value

Functions that should return registry value(s) or key(s) return them in a character string, or they return NA if the key/value is not found in the registry.

tk2reg.broadcast(), tk2reg.delete(), tk2reg.deletekey(), tk2reg.set() and tk2reg.setkey() return TRUE in case of success and FALSE otherwise.

tk2reg.get() should handle correctly the types 'sz', 'expand_sz' and multi_sz' (note that 'expand_sz' string is NOT expanded!), as well as dword' and 'dword_big_endian' that are converted into numeric values. Other types are not converted and the Tcl expression is returned ('objTcl' class) untransformed.

tk2reg.set() currently works with 'sz', 'expand_sz', 'multi_sz', dword' and 'dword_big_endian' types. A couple of other types are accepted by the function... but they are not tested ('binary', 'link', resource_list').

Note

For Windows only. These functions issue an error when they are called under other platforms. Take care while manipulating the Windows registry! You can easily lock the system completely, if you delete important items, especially if you are logged as administrator on your computer. Make a backup of your registry first before experimenting with these function!!!

Author(s)

Philippe Grosjean

See Also

tk2dde.exec(), tk2ico.create()

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

### Examples of tk2reg - registry manipulation under Windows
# Rem: HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CLASSES_ROOT, HKEY_CURRENT_USER,
#      HKEY_CURRENT_CONFIG, HKEY_PERFORMANCE_DATA, HKEY_DYN_DATA
Rkey <- "HKEY_CURRENT_USER\\\\Software\\\\R-core\\\\R"   # The R key
Rkey <- paste(Rkey, "\\\\", R.version$major, ".", R.version$minor, sep = "")
Rsubkey <- paste(Rkey, "subkey", sep = "\\\\")        # A subkey

# Get all subkeys for Software in the local machine
tk2reg.keys("HKEY_LOCAL_MACHINE\\\\Software")

# Get all names in the R key
tk2reg.values(Rkey)

# Get the path for the current R version
tk2reg.get(Rkey, "InstallPath")

# Create a subkey (explore the registry with regedit.exe to see it)
tk2reg.setkey(Rsubkey)
# Add a couple of keys in it
tk2reg.set(Rsubkey, "test", "a key added in the registry!", type = "sz")
#tk2reg.set(Rsubkey, "test exp", "\
tk2reg.set(Rsubkey, "test multi", LETTERS[1:5], type = "multi_sz")
tk2reg.set(Rsubkey, "test dword", 1024, type = "dword")
tk2reg.set(Rsubkey, "test big end", 1024, type = "dword_big_endian")

# Get the type of a value
tk2reg.type(Rsubkey, "test")
```

```
tk2reg.type(Rsubkey, "test exp")
tk2reg.type(Rsubkey, "test multi")
tk2reg.type(Rsubkey, "test dword")
tk2reg.type(Rsubkey, "test big end")

# Get a value in a key
tk2reg.get(Rsubkey, "test")
tk2reg.get(Rsubkey, "test exp")
tk2reg.get(Rsubkey, "test multi")
tk2reg.get(Rsubkey, "test dword")
tk2reg.get(Rsubkey, "test big end")

# Delete a name in a key (take care: dangerous!)
tk2reg.delete(Rsubkey, "test")
# Delete a whole key (take care: very dangerous!)
tk2reg.deletekey(Rsubkey)

# An alternate way to get the path
tk2reg.get(paste("HKEY_LOCAL_MACHINE", "SYSTEM", "CurrentControlSet",
  "Control", "Session Manager", "Environment", sep = "\\\\"), "path")

# Make sure that currently running apps are warned of your changes in the registry
tk2reg.broadcast()

# Delete temporary variables
rm(list = c("Rkey", "Rsubkey"))

## End(Not run)
```

`tk2swaplist` *A list selector that allows to select and arrange items freely*

Description

The swaplist is perfect to select and arrange items in a given order from a fixed initial set of possible items.

Usage

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

Arguments

`items`	A vector with all items.
`selection`	A vector with preselected items (must be a subset of `items`).
`title`	The title of the dialog box, by default, "Select items".
`...`	Further parameters passed to swaplist, see its tcl man page: https://core.tcl-lang.org/tklib/doc/trunk/embedded/www/tklib/files/modules/swaplist/swaplist.html.

Value

A vector with the selected items in the chosen order.

See Also

tk2listbox(), tk2tablelist()

Examples

```
## Not run:
library(tcltk2)
# tk2swaplist() makes its use super-easy
tk2swaplist(1:9, selection = c(1, 3, 5))

# Use of the swaplist on your own
tclRequire("swaplist")
tt <- tktoplevel()
opts <- tclVar()
sl <- tcl("swaplist::swaplist", tt, opts, 1:9, c(1, 3, 5))
cat("You choose:", tclvalue(opts), "\n")
rm(opts, sl, tt)

## End(Not run)
```

---

tk2tip *Display and manage tooltips in Tk widgets*

Description

tk2tip() provides a simple mechanism to display tooltips on Tk widgets when the mouse cursor hoovers 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.

Value

The current tip or NULL depending on the function.

Note

This implementation is done in pure Tcl code.

Author(s)

Philippe Grosjean

See Also

tk2button(), label()

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(tt))
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 tooltip for this widget

## End(Not run)
```

Index

∗ **utilities**
  is.tk2widget, 2
  makeTclNames, 5
  setLanguage, 7
  tclAfter, 9
  tk2button, 12
  tk2chooseFont, 17
  tk2column, 18
  tk2dde, 22
  tk2font.get, 26
  tk2ico.create, 28
  tk2reg.broadcast, 29
  tk2tip, 33

base::addTaskCallback(), *11*
base::gettextf(), *8*
base::Sys.sleep(), *11*

config *(*is.tk2widget*)*, 2
config<- *(*is.tk2widget*)*, 2

disabled *(*is.tk2widget*)*, 2
disabled<- *(*is.tk2widget*)*, 2

getLanguage *(*setLanguage*)*, 7
getLanguage(), *8*

is.tk *(*tk2column*)*, 18
is.tk(), *20*
is.tk2widget, 2
is.ttk *(*tk2column*)*, 18
is.ttk(), *14*, *20*

label *(*is.tk2widget*)*, 2
label(), *34*
label<- *(*is.tk2widget*)*, 2

makeTclNames, 5

print.tclTask *(*tclAfter*)*, 9
print.tk2cfglist *(*is.tk2widget*)*, 2
print.tk2widget *(*is.tk2widget*)*, 2

selection *(*is.tk2widget*)*, 2
selection<- *(*is.tk2widget*)*, 2
setLanguage, 7
setLanguage(), *8*
size *(*is.tk2widget*)*, 2
state *(*is.tk2widget*)*, 2

tag *(*is.tk2widget*)*, 2
tag<- *(*is.tk2widget*)*, 2
tclAfter, 9
tclAfter(), *11*
tclAfterCancel *(*tclAfter*)*, 9
tclAfterInfo *(*tclAfter*)*, 9
tclAfterInfo(), *11*
tclFun *(*makeTclNames*)*, 5
tclFun(), *11*
tclGetValue *(*makeTclNames*)*, 5
tclmc *(*setLanguage*)*, 7
tclmc(), *8*
tclmclocale *(*setLanguage*)*, 7
tclmclocale(), *8*
tclmcset *(*setLanguage*)*, 7
tclmcset(), *8*
tclSetValue *(*makeTclNames*)*, 5
tclSetValue(), *26*
tclTaskChange *(*tclAfter*)*, 9
tclTaskDelete *(*tclAfter*)*, 9
tclTaskGet *(*tclAfter*)*, 9
tclTaskGet(), *10*, *11*
tclTaskRun *(*tclAfter*)*, 9
tclTaskRun(), *11*
tclTaskSchedule *(*tclAfter*)*, 9
tcltk::tclVar(), *6*
tclVarExists *(*makeTclNames*)*, 5
tclVarFind *(*makeTclNames*)*, 5
tclVarName *(*makeTclNames*)*, 5
tip *(*tk2tip*)*, 33
tip<- *(*tk2tip*)*, 33

tk2button, 12
tk2button(), *5*, *20*, *21*, *34*
tk2canvas *(*tk2button*)*, 12
tk2cfglist *(*is.tk2widget*)*, 2
tk2checkbutton *(*tk2button*)*, 12
tk2chooseFont, 17
tk2chooseFont(), *17*, *27*
tk2column, 18
tk2column(), *20*
tk2combobox *(*tk2button*)*, 12
tk2configList *(*tk2column*)*, 18
tk2ctext *(*tk2button*)*, 12
tk2dataList *(*tk2column*)*, 18
tk2dde, 22
tk2dde(), *23*
tk2dde.exec(), *29*, *31*
tk2edit, 25
tk2edit(), *6*
tk2entry *(*tk2button*)*, 12
tk2font.get, 26
tk2font.get(), *27*
tk2font.set *(*tk2font.get*)*, 26
tk2font.set(), *27*
tk2font.setstyle *(*tk2font.get*)*, 26
tk2font.setstyle(), *27*
tk2frame *(*tk2button*)*, 12
tk2ico.create, 28
tk2ico.create(), *31*
tk2ico.destroy *(*tk2ico.create*)*, 28
tk2ico.destroy(), *28*
tk2ico.list *(*tk2ico.create*)*, 28
tk2ico.load *(*tk2ico.create*)*, 28
tk2ico.load(), *28*
tk2ico.set *(*tk2ico.create*)*, 28
tk2ico.setFromFile *(*tk2ico.create*)*, 28
tk2ico.sizes *(*tk2ico.create*)*, 28
tk2insert.multi *(*tk2column*)*, 18
tk2insert.multi(), *20*
tk2killtip *(*tk2tip*)*, 33
tk2label *(*tk2button*)*, 12
tk2labelframe *(*tk2button*)*, 12
tk2list.delete *(*tk2column*)*, 18
tk2list.get *(*tk2column*)*, 18
tk2list.insert *(*tk2column*)*, 18
tk2list.insert(), *17*
tk2list.set *(*tk2column*)*, 18
tk2list.size *(*tk2column*)*, 18
tk2listbox *(*tk2button*)*, 12
tk2listbox(), *17*, *33*
tk2mclistbox *(*tk2button*)*, 12
tk2menu *(*tk2button*)*, 12
tk2menubutton *(*tk2button*)*, 12
tk2message *(*tk2button*)*, 12
tk2notebook *(*tk2button*)*, 12
tk2notetab *(*tk2column*)*, 18
tk2notetraverse *(*tk2column*)*, 18
tk2panedwindow *(*tk2button*)*, 12
tk2progress *(*tk2button*)*, 12
tk2radiobutton *(*tk2button*)*, 12
tk2reg.broadcast, 29
tk2reg.broadcast(), *30*
tk2reg.delete *(*tk2reg.broadcast*)*, 29
tk2reg.delete(), *30*
tk2reg.deletekey *(*tk2reg.broadcast*)*, 29
tk2reg.deletekey(), *30*
tk2reg.get *(*tk2reg.broadcast*)*, 29
tk2reg.get(), *23*, *29*, *30*
tk2reg.keys *(*tk2reg.broadcast*)*, 29
tk2reg.set *(*tk2reg.broadcast*)*, 29
tk2reg.set(), *30*, *31*
tk2reg.setkey *(*tk2reg.broadcast*)*, 29
tk2reg.setkey(), *30*
tk2reg.type *(*tk2reg.broadcast*)*, 29
tk2reg.values *(*tk2reg.broadcast*)*, 29
tk2scale *(*tk2button*)*, 12
tk2scrollbar *(*tk2button*)*, 12
tk2separator *(*tk2button*)*, 12
tk2spinbox *(*tk2button*)*, 12
tk2state.set *(*tk2column*)*, 18
tk2style *(*tk2column*)*, 18
tk2swaplist, 32
tk2table *(*tk2button*)*, 12
tk2tablelist *(*tk2button*)*, 12
tk2tablelist(), *33*
tk2text *(*tk2button*)*, 12
tk2text(), *17*
tk2theme *(*tk2column*)*, 18
tk2theme.list(), *20*
tk2tip, 33
tk2tip(), *5*, *21*, *33*
tk2tree *(*tk2button*)*, 12
tk2unicode_bind *(*tk2chooseFont*)*, 17
tk2unicode_config *(*tk2chooseFont*)*, 17
tk2unicode_config(), *17*
tk2unicode_select *(*tk2chooseFont*)*, 17
tk2unicode_select(), *17*

value (is.tk2widget), 2
value<- (is.tk2widget), 2
values (is.tk2widget), 2
values<- (is.tk2widget), 2
visibleItem (is.tk2widget), 2