

Package ‘tcltk2’

July 3, 2009

Title Tcl/Tk Additions

Version 1.1-0

Date 2009-07-02

Depends R (>= 2.4.0), tcltk

Suggests utils

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

Author Philippe Grosjean <phgrosjean@sciviews.org>

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.

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

License file LICENSE

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

Repository CRAN

Date/Publication 2009-07-03 07:19:04

R topics documented:

tcltk2-package	2
setLanguage	2
tclTask	3
tclVarFun	5
tk2commands	7
tk2dialogs	9
tk2edit	10
tk2fonts	11
tk2tip	12
tk2widgets	13

Index	17
--------------	-----------

tcltk2-package *Tcl/Tk Additions*

Description

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

Details

Package: tcltk2
Type: Package
Version: 1.0-9
Date: 2009-06-27
License: 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 determine 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 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

```
# 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 <code>tclTaskGet()</code> , retrieve the list of all existing tasks.
all	If <code>id = NULL</code> , <code>all = TRUE</code> indicate to list all tasks, including hidden ones (with id starting with a dot).
redo	Should the task be rescheduled n times, indefinitely (<code>redo = TRUE</code>) or not (<code>redo = FALSE</code> , 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 exist, `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 ====\\n"
tclTaskSchedule(1000, test3(msg), id = "test3", redo = TRUE)
Sys.sleep(2)
msg <- "==== Second message ====\\n"
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)
```

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 functionalities 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 not 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()...
# intended 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)

```

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 (TclObjobject)
tab	The name (text) of a tab in a notebook
state	The new state of the widget
theme	A theme to use (character string)

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 adventurous, you can even design your own themes (see the tile documentation on the Tcl wiki).

Author(s)

Philippe Grosjean

See Also

[tk2widgets](#), [tk2tip](#)

tk2dialogs

Additional Tk dialog boxes

Description

Tk dialog boxes to select a font, a R color, etc.

Usage

```
tk2chooseFont(...)
```

Arguments

... Further arguments passed to the dialog box

Value

The selection made in the dialog box if OK is clicked, "" otherwise.

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

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")
## 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-effect, 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 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 .fontStyleXXX in 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)
```

tk2tip

Display and manage tooltips in Tk widgets

Description

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

Usage

```
tk2tip(widget, message)
tk2killtip()
```

Arguments

<code>widget</code>	The widget to which a tooltip is attached
<code>message</code>	The message of the tooltip ("" to remove the tooltip)

Note

This implementation is done in pure Tcl code

Author(s)

Philippe Grosjean

See Also[tk2widgets](#)**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)
tt2 <- tkoplevel()
lb <- tklabel(tt2,
             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(tt2, 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.
## 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 = "", ...)
tk2checkboxbutton(parent, tip = "", ...)
tk2combobox(parent, tip = "", ...)
tk2entry(parent, tip = "", ...)
tk2frame(parent, ...)
tk2label(parent, tip = "", ...)
tk2labelframe(parent, ...)
tk2listbox(parent, selectmode = c("single", "browse", "multiple", "extended"),
           tip = "", ...)
tk2mclistbox(parent, tip = "", ...)
tk2menu(parent, ...)
```

```

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, ...)
tk2text(parent, tip = "", ...)
tk2ctext(parent, tip = "", ...)
tk2tree(parent, tip = "", ...)

```

Arguments

parent	The parent window
tip	A tooltip to display for this widget
selectmode	The selection mode for this widget
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 large, 50 = twice taller). Only used if width is negative
orientation	Either "horizontal" or "vertical"
...	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)
#### TO DO: 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)
## End(Not run)

```

Index

*Topic **package**

tcltk2-package, 1

*Topic **utilities**

setLanguage, 2

tclTask, 3

tclVarFun, 5

tk2commands, 7

tk2dialogs, 9

tk2edit, 10

tk2fonts, 11

tk2tip, 12

tk2widgets, 13

addTaskCallback, 4

getLanguage (*setLanguage*), 2

is.tk (*tk2commands*), 7

is.ttk, 14

is.ttk (*tk2commands*), 7

makeTclNames (*tclVarFun*), 5

setLanguage, 2

Sys.sleep, 4

tclAfter (*tclTask*), 3

tclAfterCancel (*tclTask*), 3

tclAfterInfo (*tclTask*), 3

tclFun, 2, 4

tclFun (*tclVarFun*), 5

tclGetValue (*tclVarFun*), 5

tclSetValue, 10

tclSetValue (*tclVarFun*), 5

tclTask, 3

tclTaskChange (*tclTask*), 3

tclTaskDelete (*tclTask*), 3

tclTaskGet (*tclTask*), 3

tclTaskRun (*tclTask*), 3

tclTaskSchedule (*tclTask*), 3

tcltk2 (*tcltk2-package*), 1

tcltk2-package, 1

tclVarExists (*tclVarFun*), 5

tclVarFind (*tclVarFun*), 5

tclVarFun, 5

tclVarName (*tclVarFun*), 5

tk2button (*tk2widgets*), 13

tk2canvas (*tk2widgets*), 13

tk2checkbutton (*tk2widgets*), 13

tk2chooseFont (*tk2dialogs*), 9

tk2column (*tk2commands*), 7

tk2combobox (*tk2widgets*), 13

tk2commands, 7, 9

tk2ctext (*tk2widgets*), 13

tk2dialogs, 9

tk2edit, 6, 10

tk2entry (*tk2widgets*), 13

tk2font.get (*tk2fonts*), 11

tk2font.set (*tk2fonts*), 11

tk2font.setstyle (*tk2fonts*), 11

tk2fonts, 11

tk2frame (*tk2widgets*), 13

tk2insert.multi (*tk2commands*), 7

tk2killtip (*tk2tip*), 12

tk2label (*tk2widgets*), 13

tk2labelframe (*tk2widgets*), 13

tk2list.delete (*tk2commands*), 7

tk2list.get (*tk2commands*), 7

tk2list.insert (*tk2commands*), 7

tk2list.set (*tk2commands*), 7

tk2list.size (*tk2commands*), 7

tk2listbox (*tk2widgets*), 13

tk2mclistbox (*tk2widgets*), 13

tk2menu (*tk2widgets*), 13

tk2menubutton (*tk2widgets*), 13

tk2message (*tk2widgets*), 13

tk2notebook (*tk2widgets*), 13

tk2notetab (*tk2commands*), 7

tk2notetraverse (*tk2commands*), 7

tk2panedwindow (*tk2widgets*), 13

`tk2progress` (*tk2widgets*), 13
`tk2radiobutton` (*tk2widgets*), 13
`tk2scale` (*tk2widgets*), 13
`tk2scrollbar` (*tk2widgets*), 13
`tk2separator` (*tk2widgets*), 13
`tk2spinbox` (*tk2widgets*), 13
`tk2state.set` (*tk2commands*), 7
`tk2table` (*tk2widgets*), 13
`tk2text` (*tk2widgets*), 13
`tk2theme` (*tk2commands*), 7
`tk2tip`, 8, 12
`tk2tree` (*tk2widgets*), 13
`tk2widgets`, 2, 8, 9, 12, 13