Package 'fgui'

February 19, 2015

Version 1.0-5

Date 2009-04-16

Title Function GUI

Author Thomas Hoffmann <tjhoffm@gmail.com>

Maintainer Thomas Hoffmann <tjhoffm@gmail.com>

Imports tcltk

Suggests tcltk

Depends tools

Description Rapidly create a GUI interface for a function you created by automatically creating widgets for arguments of the function. Automatically parses help routines for context-sensative help to these arguments. The interface essentially a wrapper to some tcltk routines to both simplify and facilitate GUI creation. More advanced tcltk routines/GUI objects can be incorporated into the interface for greater customization for the more experienced.

License GPL

URL http://www.people.fas.harvard.edu/~tjhoffm/fgui.html

LazyLoad true

Repository CRAN

Date/Publication 2012-12-27 19:29:17

NeedsCompilation no

R topics documented:

gui 2
guiFrame 11
guiGetValue 13

Index **15**

gui *fgui Main Function*

Description

Rapidly create a GUI interface for a function you created by automatically creating widgets for arguments of the function. Automatically parses help routines for context-sensative help to these arguments. The interface is essentially a wrapper to some tcltk routines to both simplify and facilitate GUI creation. More advanced tcltk routines/GUI objects can be incorporated into the interface for greater customization for the more experienced.

The examples are probably the quickest/easiest way of understanding what this code does.

`gui` and `guiv` are the main routines. The latter returns the value of the function. The former returns the list of arguments chosen by the user, not the value of the function (which would need to be computed separately, see the code in `guiv` to see how to do so).

Other main but more advanced functions are as follows. `guiNestedF` providing a means for nested forms (see examples). Along with this is `guiFormals`, a modification of the `formals` routine to deal with issues.

Use `getFromNamespace` and `assignInNamespace` to customize any of the widget drawing routines (described elsewhere) that are not to your tastes, or alter the source code (get from CRAN so you get one with comments).

`guiExec` provides the means for 'immediate' updates, e.g. for power interfaces; see examples for exactly how to do this.

On the next level, `mgui` creates a menuing interface when you want to have seperate routines to create multiple graphical widgets. For example, when converting multiple command line routines in an R package. Just use `mgui` instead of `fgui`, for the most part – but if you want the interface modal, only do so on the last mgui call. The functions `fguiNewMenu`, `fguiWindow`, and `fguiWindowPrint` provide lower level workings for this. `fguiNewMenu` allows for more general menu additions. `fguiWindow` allows for more options in creating the main window (otherwise it is automatically created on the first `mgui` call). `fguiWindowPrint` prints directly to the console, although many times this is also caught by other routines.

Usage

```
gui( func,
      argOption=NULL, argFilename=NULL, argList=NULL, argSlider=NULL,
      argCommand=NULL, argEdit=NULL, argFilter=NULL,
      argText=NULL, argType=NULL,
      argGridOrder=1:length(formals(func)),
        argGridSticky=rep("a",length(formals(func))),
        argGridFrame=rep("f",length(formals(func))),
      title=NULL,
      exec="OK",
      closeOnExec=is.null(output), cancelButton=TRUE,
      callback=NULL,
      output='m',
```

```
      helps='auto', helpsFunc=NULL,
      grid=TRUE, modal=NULL, nameFix=TRUE, getFix=TRUE,
      verbose=FALSE )
guiv( func=NULL, output=NULL, modal=TRUE, title=NULL, ... )

guiNestedF( func, nestArg, title=nestArg, exec=NULL, output=NULL, ... )
guiExec( lastTouched=NULL )
guiFormals( func, object )

mgui( func,
      argOption=NULL, argFilename=NULL, argList=NULL, argSlider=NULL,
      argCommand=NULL, argEdit=NULL, argFilter=NULL,
      argText=NULL, argType=NULL,
      argGridOrder=1:length(formals(func)),
        argGridSticky=rep("a",length(formals(func))),
        argGridFrame=rep("f",length(formals(func))),
      title=NULL,
      exec="OK",
      closeOnExec=is.null(output), cancelButton=TRUE,
      callback=NULL,
      output='m',
      helps='auto', helpsFunc=NULL,
      grid=TRUE, modal=TRUE, nameFix=TRUE, getFix=TRUE,
      verbose=FALSE )
fguiNewMenu( menuText, command=function(){print(paste(menuText,collapse=" > "))} )
fguiWindow( basicMenu=TRUE, title="fgui", text="Please choose an option from the menu." )
fguiWindowPrint( text, endl=TRUE )
```

Arguments

func	The function that should be called upon execution. The one required argument.
argOption	list of options vectors (names should be the same as args of provided function 'func', makes the arg become an option widget)
argList	list of strings for lists (names should be the same as args), makes arg be a list widget, which can be customized on the fly (an option cannot be, and has limited size)
argSlider	list of slider ranges (names should be the same as args), makes arg be a slider/range
argCommand	list of functions to execute on command (names same as args), makes arg be a command button
argEdit	list of (width,height) both optional (names same as args), NULL/NA/missing for default, makes arg be an edit box
argFilter	list of file filters (empty for all files, names same as args), makes arg be a file choosing widget
argFilename	list of default filenames (names same as args), makes arg be a file choosing widget
argText	list of default text for text boxes (names same as args), makes arg be a text box (which is the default anyway).

argGridOrder	Order to be gridded; if two objects have the same order, they will be gridded side by side.
argGridSticky	Vector of 'sticky' values for each component to be gridded. Each sticky value is a character string with values in n=north, s=south, e=east, w=west. So "nws" would make it take the entire vertical space and be on the west.
argGridFrame	A vector of values for each unique grid order. The value 'f' is for creating a frame to enclose all of the components within, and 'g' for grids directly. The former looks good with mixed types of components, but the latter will look better for aligning components in columns.
title	Window title, defaults to function name.
argType	Unspecified is auto-detected, and is strongly recommended (only necessary for 'i'). List of the types of each of your arguments, with the name being the same as the argument name. 't': text entry. 's': slider. 'f': input for filenames. 'o': options box (options are put in argOption). 'l': list box (lists are put in argList, which is 'set', and can be modified by user). 'c': command button. 'm': multi-line text entry. 'i': ignore - option will not be drawn, and the default will be used [not fully tested, try creating a seperate function and using the helpsFunc option].
exec	String to use when user should press a button to have them call your function. Empty indicates it should not be drawn (e.g. power interfaces, where you might desire sliders, and an auto-updating answer).
closeOnExec	Whether to close when the 'OK' (default text, but this can be changed) button is pressed.
cancelButton	Whether to include a button that allows the user to cancel execution of the function.
callback	Name of function to handle callbacks, that takes one parameter, a string for the arg that was updated.
output	one of the above, 't', 's', 'm', or NULL; will try to auto-choose this as well. If not 'm', then an initial value will be set by running the default parameters.
helps	'auto' indicates it will try to load in the help from the package help, if possible. Otherwise this can be a list of strings (with the function argument name for the names) for help.
helpsFunc	NULL indicates it will use the name of the function. If a string is provided, it will try to load the help on the function specified by that string instead.
grid	whether to grid the objects or not (otherwise, just let the user do it)
modal	lock input away from R, suggested
nameFix	boolean, tries to fix names (replaces underscore and period with a space).
getFix	boolean, tries to fix strings that represent R objects - so for instance, if a user wanted to user the dataset 'rivers', they would only need to type 'rivers' (without the quotation marks) to represent the dataset.
verbose	Prints out verbose output. Only useful to try to understand what it is doing if you are wanting to customize.
nestArg	Name of the argument to be nested, see examples.

...	Options to nestArg that are passed on to the gui function, so everything you see above.
lastTouched	Not used. Required to have conformity with callback routines.
object	See below, only for nested forms.
menuText	An array of strings representing the menu. E.g. c("File","Exit").
command	Command to be run on click.
basicMenu	Inserts a basic menu (File, etc.).
text	Text in the text box for the gui choosing interface.
endl	Whether to insert a carriage return.

Details

Examples are strong recommended to get the idea of what this does. The reference below is a great one for learning how to use tcltk.

Note that NULL, NA, and NaN support is limited to the text entry types only, and may perform unexpectedly for other types.

Value

guiv returns the function evaluated at the list of arguments chosen by the user. gui returns only the list of the arguments the user has chosen.

When run in modal mode, the values are returned. When not in modal form, the values may be accessed with guiv() and guiGetAllValues.

References

Dalgard, Peter and Wettenhall, James. R tcltk examles. http://www.sciviews.org/_rgui/tcltk/index.html

Examples

```
## Not run:
##################################################
## *** EXAMPLE 1 ***                            ##
## Basic example of available graphical objects ##

## our function to base the GUI on
demofunc <- function( opt, lst, slide, cmd, ed, txt, flname ) {
  ## Returns a string of output, this will be displayed
  return( paste( "opt:",    opt,
                 "lst:",    paste(lst,collapse=","),
                 "slide:",  slide,
                 "ed:",     ed,
                 "txt:",    txt,
                 "flname:", flname,
                 sep="\n" ) )
}
```

```
## Simple callback example
cmdCallback <- function() {
  tkmessageBox( message="Hello World :)", title="A Classic" )
}
## start the gui
res <- gui( demofunc,
          argOption=list(opt=c("TRUE","FALSE")), ## names in list are that of args in func
            argList=list(lst=c(as.character(1:10))),
            argSlider=list(slide=c(0,100,2.5)),   ## start,stop,stepsize
            argCommand=list(cmd=cmdCallback),
            argEdit=list(ed=NULL), ## otherwise (width,height) to tweak, default
          argFilter=list(flname="{{Text files} {.txt}}") )  ## note space inbetween the braces!
## prints out the arguments the user chose
print( res )

## End(Not run)

## Not run:
########################
## ** EXAMPLE 2 ***   ##
## Auto-loading help! ##
## This is extremely useful if you write your own R package
##  and want to include help with the GUI with no fuss.

## This is what this looks like
help("rnorm")
## Now build a gui
gui( rnorm )

## Now, suppose we wanted to customize it,
##  but we really want to keep all that help...
rnorm2 <- function( n=10, mean=1, sd=2 ) {
  res <- rnorm( n=n, mean=mean, sd=sd )
  return( paste( res, collapse=", " ) )
}
gui( rnorm2, helpsFunc="rnorm" )

## End(Not run)

## Not run:
######################
## ** EXAMPLE 3 *** ##
## Power interface  ##

ss <- function( alpha=0.05, beta=0.8, sigma=2, effect_size=0.5 ) {
  n <- ceiling(  (qnorm(1-alpha/2) + qnorm(1-beta))^2 * sigma^2 / effect_size^2  )
  print(n)
  return(n)
}
## Create the gui
## Note 1: the use of output in the slider
## Note 2: callback set to the 'guiExec' (fixed) routine,
##          so 'ss' is run with the proper arguments
```

```
##          whenever a slider value is changed
gui( ss,
     argSlider=list(alpha=c(0,0.1,0.001),
                    beta=c(0,1,0.01),
                    sigma=c(0,10),
                    effect_size=c(0,10),
                    output=c(0,10000,1)), ## Note the use of output here
     exec=NULL, ## don't draw an execute button
     callback=guiExec
    )

## End(Not run)

## Not run:
################################
## *** Example 4 ***           ##
## Sliders setting each other. ##
## You can envision this for more complicated power interfaces
##  that do both calculating power and solving for sample
##  sizes...
## Also includes non-auto help, a waste to bother with
##  if you are planning on creating a package

## Change a default for fun, see `guiSet' function
##  for more details/options
guiSet( "SLIDER_LENGTH", 400 )


sli <- function( alpha=0.5, beta=0.5 ) {
  ## Nothing to do...
}
sliCallback <- function( lastTouched ) {
  if( lastTouched=="alpha" )
    guiSetValue("beta",guiGetValue("alpha"))  ## setting beta to be alpha
  if( lastTouched=="beta" )
    guiSetValue("alpha",guiGetValue("beta"))  ## setting alpha to be beta
}
gui( sli,
     argSlider=list(alpha=c(0,1), beta=c(0,1)),
     output=NULL, exec=NULL, callback=sliCallback,
     helps=list(alpha="type I error", beta="power") )

## End(Not run)

## Not run:
################################
## *** EXAMPLE 5 ***         ##
## Parsing R objects example ##
## Suppose you want a user to be able to enter a vector of data,
##  then you can use the following as an example for that.
summaryStats <- function( data ) {
  return( paste( "Mean = ", mean(data), ", Variance = ", var(data), sep="" ) )
}
```

```
gui( summaryStats, helps=list(data="Enter the data as an R vector, e.g. 'c(13,66,44,27)' or enter the text 'river

## End(Not run)

## Not run:
################################
## *** EXAMPLE 6 ***         ##
## Advanced nesting example. ##
## Suppose we have a function 'f', which has too many
##  arguments to comfortably fit on one screen.
f <- function( a=1, b=2, c=3, d=4, e=5, f=6 ) {
  print( "Running f" )
  return( paste( "a =", a, "\n",
                 "b =", b, "\n",
                 "c =", c, "\n",
                 "d =", d, "\n",
                 "e =", e, "\n",
                 "f =", f, "\n", sep="" ) )
}
## Say we split into two functions/forms
f1 <- function( a=1, b=2, c=3 ) {
  print( "Running f1" )
  return( list(a=a,b=b,c=c) )
}
f2 <- function( d=4, e=5, f=6 ) {
  print( "Running f2" )
  return( list(d=d,e=e,f=f) )
}
## Then our main gui function could be
guif <- function( abc, def ) {
  print( "guif" )

  print( "guif: abc" )
  print( abc )
  print( "guif: def" )
  print( def )

  f <- guiFormals( f, c(abc,def) )
  f()
}
gui( guif, argCommand=list(abc=guiNestedF(f1,"abc"), def=guiNestedF(f2,"def")) )

## End(Not run)

################################
## *** EXAMPLE 7 ***         ##
## The menuing interface.    ##
## Call just as you would gui, same options, same everything,
##  EXCEPT title is now a vector indicating the menu path.
## If you want it modal though, do not do so until the last mgui call, or it will be modal inbetween additions to t
## Not run:
fguiWindowPrint( "Goes to the console because no window has been created." )
```

```
mgui( rgeom, title=c("Random","Geometric") )
mgui( rbinom, title=c("Random","Binomial") )
fguiNewMenu( c("Random","SEPARATOR") ) ## Puts a separator in the menu
mgui( rnorm, title=c("Random","Normal") )
mgui( runif, title=c("Random","Uniform") )
fguiWindowPrint( "Goes to the main window, now that it has been created." )

## End(Not run)

################################
## *** EXAMPLE 8 ***         ##
## Basic lm() interface.     ##
## Not run:
lmgui <- function( csvFilename, response, explanatory ) {
  ## Construct a formula for the 'lm' routine
  modelStr <- paste( response, "~", paste( explanatory, collapse="+" ) )
  ## Load in the data
  data <- read.csv( csvFilename )
  ## perform the regression, give the summary
  return <- summary(  lm( formula=modelStr, data=data )  )
}

lmguiCallback <- function( arg ) {
  if( arg=="csvFilename" ) {
    ## A dataset was chosen
    ## - The filename corresponds to the value at that argument
    ## - So pull of the names of that dataset
    datanames <- names( read.csv( guiGetValue("csvFilename") ) )
    print( datanames )
    ## - Store the datanames for future use, think of this as a global variable
    guiSet( "datanames", datanames )
    ## - Set the possible values for the response
    setListElements( "response", datanames )
    setListElements( "explanatory", datanames )
  }
}

guiv( lmgui, argFilename=list(csvFilename=NULL), argList=list(response=NULL,explanatory=NULL), callback=lmguiCa

## End(Not run)
################################
## *** EXAMPLE 9 ***         ##
## Advanced lm() interface.  ##
## Not run:
## The function we will pass to guiv is somewhat of a shell here, that is it would not
##  make sense to use it from the command line. It's specification
##  is only to create a GUI using fgui.
lmgui2 <- function( csvFilename,   ## Create file dialogue, special callback
                    simData,       ## Only for a command button
                    response,      ## Required input
                    explanatory,   ## Required input
                    scatter,       ## Only for a command button
                    summary ) {    ## Only for a command button
```

```
  ## Data has been loaded in callback routine,
  ##  into what can be thought of as a global variable
  data <- guiGetSafe("PERSONAL_dataset")
  if( class(data)[1] != "data.frame" )
    stop("Data must be loaded.") ## Gives error message box

  ## Error check: response and explanatory should have been set
  if( length(response)==0 ) stop( "Must specify a response." )
  if( length(explanatory)==0 ) stop( "Explanatory variable expected." )

  ## Run and return the fit from 'lm' linear model
  modelStr <- paste( response, "~", paste( explanatory, collapse="+" ) )
  return( lm( formula=modelStr, data=data ) )
}
lmgui2Callback <- function( arg ) {
  if( arg=="csvFilename" ) {
    ## Dataset chosen from file dialogue,
    ##  so we should load it in.
    data <- read.csv( guiGetValue("csvFilename") )
    guiSet( "PERSONAL_dataset", data ) ## think of as a global variable
    ## Also set possible values for response and explanatory variables
    setListElements( "response", names(data) )
    setListElements( "explanatory", names(data) )
  }else if( arg=="simData" ) {
    ## Generate a random set of data, and write to disk
    set.seed(13); library(MASS);
   data <- data.frame( mvrnorm( n=100, mu=c(0,0,0), Sigma=matrix(c(1,0.3,0, 0.3,1,0.3, 0,0.3,1),nrow=3) ) )
    names( data ) <- c("Response","Covariate1","Covariate2")
    write.csv( data, "lmgui2_generated.csv", row.names=FALSE )
    ## Now set it as if it was loaded in, and run that callback
    guiSetValue( "csvFilename", "lmgui2_generated.csv" )
    lmgui2Callback( "csvFilename" )
  }else if( arg=="scatter" ) {
    ## Create a scatterplot of everything in the dataset
    data <- guiGetSafe("PERSONAL_dataset")
    response <- guiGetValue("response")
    wh.response <- which(names(data)==response)
    if( length(wh.response) != 1 )
      stop( "One and only one response must be chosen." )
    if( class(data)[1] != "data.frame" )
      stop( "Data must be loaded." )
    par( mfrow=rep( ceiling(sqrt(ncol(data)-1)), 2 ) )
    for( i in setdiff(1:ncol(data),wh.response) )
      plot( data[[i]], data[[wh.response]],
            xlab=names(data)[i], ylab=names(data)[wh.response] )
  }else if( arg=="summary" ) {
    print( summary( guiExec() ) ) ## when no output, guiExec returns value
  }
}
fit <- guiv( lmgui2,
             argFilename=list(csvFilename=NULL),
             argList=list(response=NULL,explanatory=NULL),
             argCommand=list(simData=NULL, scatter=NULL, summary=NULL),
```

```
                callback=lmgui2Callback,
                argGridOrder=c(1,1,2,2,3,3), ## Multi-column ordering
                argText=c(csvFilename="Load data (csv)",
                          simData="Simulate data",
                          response="Choose response variable",
                          explanatory="Choose explanatory variable",
                          scatter="Generate scatterplot to response variable",
                          summary="Print summary")
      )

## End(Not run)
```

---

guiFrame *Various Graphical Widgets*

Description

Creates various graphical objects, used internally by the 'gui' routine, but exported in case useful and as examples of further customization. Try `getFromNamespace` and `assignInNamespace` to fully customize these routines in your own code. Documentation in tcltk may also help.

`guiFrame` creates a frame.

`guiTextEntry` creates a short one-line text entry.

`guiSlider` creates a slider with a range of values to choose, useful for power calcs.

`guiFilename` provides a means to get filenames.

`guiOption` allows a choice of options.

`guiList` allows a choice of greater options, which can be modified later. `setListElements` and `getSelectedListElements` are routines to dynamically set these lists and get the selected elements.

`guiEdit` is an edit box.

`helpButton` creates a helpButton.

Usage

```
guiFrame( sframe, grid=FALSE, relief="groove",
           borderwidth=2, sticky="nws" )

guiTextEntry( sframe, text, default, width=NULL, helps=NULL )
guiSlider( sframe, text, default, min, max, step=(max-min)/100,
            update=NULL, state="enabled", helps=NULL )
guiFilename( sframe, text="Filename ...", default="", title="",
              filter="{{All files} {.*}}", callback=NULL, helps=NULL )
guiOption( sframe, text, choices, defaultChoice=1,
            update=NULL, helps=NULL )
guiList( sframe, text, name=text, update=NULL, helps=NULL )
```

```
guiEdit( sframe, text="", default="", width=NULL, height=NULL,
          readonly=FALSE, helps=NULL )
helpButton( sframe, helps, title )

getSelectedListElements( name )
setListElements( name, elements )
```

Arguments

sframe	The tkframe to grid upon.
grid	Whether the object should be gridded or not. Default is FALSE, so user can grid objects together.
relief	tkframe option
borderwidth	tkframe option
sticky	Combination of 'nwes' for stickiness of object.
text	The text of the object, to describe it to the user.
default	Default value for an object.
min	min of slider range
max	max of slider range
step	stepsize of slider range
update	function to send callbacks to, should take one argument (see examples in 'gui' routine)
state	e.g. 'readonly', see tcltk docs
helps	An optional string of help to be given when a user clicks the '?' button to provide more information. If NULL, no such button is drawn.
title	Title for the window.
filter	File filter, see examples in 'gui' for the form.
callback	Callback function. For command function, both 'callback' and 'update' functions are performed.
choices	List of possible choices.
defaultChoice	Default choice to choose, the numeric index to this.
name	Identifier.
width	Width of the edit box.
height	Height of the edit box.
readonly	Whether to create in readonly state.
elements	Array of strings for the list elements.

Details

tkFrame and helpButton return the tcltk object reference.

The other routines return a list. The first object, object, is either a tclVar or the string 'no object' for things like command buttons where this does not make sense (use main<-tktoplevel(); res<-tkFrame(main); tclvalue
e.g., to get the value). The second object, guiObject returns a reference to the tcltk frame gui the object is contained in for gridding purposes.

Examples

```
## Not run:
## Create a form with tcltk routines
main <- tktoplevel()
## Create some widgets for that form
## - Create a frame, and put two widgets in it
## - Note that guiTextEntry objects will be gridded automatically
##   (which is why as an example they are put in a frame)
fr <- guiFrame( sframe=main )
te1 <- guiTextEntry( sframe=fr, text="Text entry 1", default="default" )
te2 <- guiTextEntry( sframe=fr, text="Text entry 2", default="" )
## - Put the rest of the widgets on the main frame
sl <- guiSlider( sframe=main, text="Slider", default=5, min=1, max=10 )
fl <- guiFilename( sframe=main, text="Filename", default="foo.txt" )
op <- guiOption( sframe=main, text="Option", choices=c("one","two","three") )
ed <- guiEdit( sframe=main, text="Edit", default="Edit box" )
## Now grid the widgets on the main form
tkgrid( fr )
tkgrid.configure( fr, sticky="nws" ) ## Handle alignment, as in tcl/tk package
tkgrid( sl$guiObject )
tkgrid( fl$guiObject )
tkgrid( op$guiObject )
tkgrid( ed$guiObject )
print( tclvalue(fl$object) ) ## will print out "foo.txt", unless modified

## End(Not run)
```

`guiGetValue` *Getting and Setting Values*

Description

`guiGetValue` and `guiGetAllValues` are used for getting the values of the objects that you created. This is useful to make more customized responses to user inputs.

The guiGet and guiSet routines get at more internal code to the interface. In particular guiSet can set some internal constants.

Usage

```
guiGetValue( i )
guiGetAllValues()
guiSetValue( i, value )

guiSet( x, value )
guiGet( x, mode="any", ifnotfound=NA )
guiGetSafe( x, ifnotfound=NA )
```

Arguments

i	Which item in the list to return. If a string, the name of the corresponding arg. If numeric, the index to the arg.
x	String to represent the object. See examples below for constants.
mode	See mode; returns results corresponding only to a certain type of object, such as numeric.
ifnotfound	Value to return if not found.
value	Value to set in the namespace.

Details

guiGetAllValues returns a list of all values of objects created, versus just one specific value. Values are returned as strings, or numeric, depending on the value (it attemps to convert everything to numeric, on failure, returns the string).

guiSet can be used to set values.

Examples

```
## Not run:
values <- guiGetAllValues()
value1 <- guiGetValue(1)

## End(Not run)

## Not run:
## These are the constants that you can modify
##  to change the way things are displayed.
guiSet( "SLIDER_LENGTH", 500 )
guiSet( "ENTRY_WIDTH", 40 )
guiSet( "LIST_HEIGHT", 15 )
guiSet( "LIST_WIDTH", 15 )
guiSet( "EDIT_WIDTH", 65 )
guiSet( "EDIT_HEIGHT", 5 )

## End(Not run)
```

Index

∗Topic **interface**
gui, 2
guiFrame, 11
guiGetValue, 13

fguiNewMenu *(*gui*)*, 2
fguiWindow *(*gui*)*, 2
fguiWindowPrint *(*gui*)*, 2

getSelectedListElements *(*guiFrame*)*, 11
gui, 2
guiEdit *(*guiFrame*)*, 11
guiExec *(*gui*)*, 2
guiFilename *(*guiFrame*)*, 11
guiFormals *(*gui*)*, 2
guiFrame, 11
guiGet *(*guiGetValue*)*, 13
guiGetAllValues *(*guiGetValue*)*, 13
guiGetSafe *(*guiGetValue*)*, 13
guiGetValue, 13
guiList *(*guiFrame*)*, 11
guiNestedF *(*gui*)*, 2
guiOption *(*guiFrame*)*, 11
guiSet *(*guiGetValue*)*, 13
guiSetValue *(*guiGetValue*)*, 13
guiSlider *(*guiFrame*)*, 11
guiTextEntry *(*guiFrame*)*, 11
guiv *(*gui*)*, 2

helpButton *(*guiFrame*)*, 11

mgui *(*gui*)*, 2

setListElements *(*guiFrame*)*, 11