Package 'tkWidgets'

November 12, 2026

Version 1.90.0

Title R based tk widgets

Author J. Zhang <jzhang@jimmy.harvard.edu>

Maintainer J. Zhang <jzhang@jimmy.harvard.edu>

Depends R (>= 2.0.0), methods, widgetTools (>= 1.1.7), DynDoc (>= 1.3.0), tools

Suggests Biobase, hgu95av2

Description Widgets to provide user interfaces. tcltk should have been installed for the widgets to run.

LazyLoad yes

biocViews Infrastructure

Keyword graphics

License Artistic-2.0

git_url https://git.bioconductor.org/packages/tkWidgets

git_branch RELEASE_3_23

git_last_commit 82dbb93

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

appendSepDir 2
args2XML 3
argsWidget 4
colInfo-class 5
dataViewer 6
dbArgsWidget 7
DPExplorer 8
eExplorer 9

fileBrowser 10
fileWizard 12
getLightTW 13
getWvalues 14
guess.sep 16
hasChar 17
importPhenoData 18
importWizard 20
listSelect 23
objectBrowser 24
objNameToList 25
objViewer 26
pExplorer 27
pickFiles 29
pickItems 30
pickObjs 31
setArgsList 32
stdType 34
tkMIAME 35
tkphenoData 37
tkSampleNames 37
values.Widget 38
vExplorer 40
widgetRender 41
WName 43

Index **45**

`appendSepDir` *List File and Directories for a Given Path*

Description

Takes a path and returns a vector of string with the names of files and directories for the path. The directory names will have a system dependent path separator (e. g. / for Unix) appended.

Usage

```
appendSepDir(path)
```

Arguments

`path`	`path` character string specifying the path whose contents are to be listed

Value

A character vector containing file and directory names.

Author(s)

Jianhua (John) Zhang

See Also

fileBrowser, pickFiles

Examples

```
# File names and directory names are not differentiated
list.files()

# Put a separater at the end of directory names
appendSepDir(getwd())
```

`args2XML` *Converting the formal arguments to a function and converts into an XML format*

Description

This function reads the formal arguments to a given function and converts the content into an XML format

Usage

```
args2XML(fun, xml.name = "", full.names = NULL, priority = NULL)
```

Arguments

`fun`	`fun` name of the function of interests
`xml.name`	`xml.name` a character string for the name of the xml file holding the content of the formal arguments to the function
`full.names`	`full.names` a vector of character string with full description of each of the formal arguments to the function. The order of apperance of each description much correspond to the oreder of their appeerance in the formal argument list
`priority`	`priority` a vector of integers or character strings indicating the priority of the arguments.

Details

Priority values are currently used to determine whether the argument will appear on a widget that has entry boxes for modifying the values of the arguments. Users of args2XML may not have any concern of the priority values

Value

No value will be returned.

Author(s)

Jianhua Zhang

See Also

fileWizard

Examples

```
fullNames <- c("Full path names", "Pattern to match",
"Visiable file names", "Include path")
args2XML(list.files, "temp.xml", fullNames, c(1, 2, 2, 2))
readLines("temp.xml")
unlink("temp.xml")
```

---

argsWidget *Functions to construct a widget that takes inputs from users*

Description

Given a argument list, the functions construct a widget to allow users to entry or select values for arguments defined by the names of the argument list.

Usage

```
argsWidget(argsList, defaultNames, inst = "")
getPWidget(argsList, PWEnv, inst = "")
formatArg(toFormat)
getSymbol(args)
funcs2Char(args,funcs)
getTrueNullNa(toFormat)
```

Arguments

argsList	argsList a list of arguments with or without default values. The list can be derived from e.g. formals
PWEnv	PWEnv an R environment used object to store data for the argument list
toFormat	toFormat element to be formated by formatArg
args	args arguments to be formated
funcs	funcs a list containing the arguments that are functions
defaultNames	defaultNames a vector of character strings of length 2 for two default buttons to be rendered. The first one is to end the process and the second to abort the process
inst	inst a character string for a short instuction that will appear on the top of a widget

Details

argsWidget builds a widget with widget elements to allow users to input values for the arguments.

getPWidget instatiates primary widgets that will be used to construct the widget for argements.

formatArg formats the value for a given argument so that the calue can be displayed by a tcltk widget.

getSymbol filters out functions from the argument list.

funcs2Char converts functions to character representations of the functions.

getTrueNullNa converts string "true", "false", "null", and "na" to R primitives for these items.

Value

argsWidget returns a list with user input values for elements of the argument list passed.

getPWidget returns a list of primary widgets.

formatArg returns a list containing the formated values.

getSymbol returns a list containing arguments that are functions.

funcs2Char returns a list containing character representations of functions.

getTrueNullNa returns an R object.

Author(s)

Jianhua Zhang

References

R tcltk

Examples

```
if(interactive()){
    argsWidget(list("Entry with default" = "default",
                    "Entry without default" = ""))
}
```

---

colInfo-class *Class "colInfo" presents column information for a data frame*

Description

This class is for keeping information about a data frame to be processed. The class is mainly for use by importWizard

Objects from the Class

Objects can be created by calls of the form new("colInfo", ...)

Slots

`colName`: Object of class `"character"` - a character string for the name of the column

`colType`: Object of class `"character"` - a character string for the data type of the column. Can only be "character" or "numeric"

`dropOrNot`: Object of class `"logical"` - a boolean indicationg whether the column will be droped

Methods

colName `signature(object = "character")`: The get method for slot "colName"

colName<- `signature(object = "character")`: The set method for slot "colName"

colType `signature(object = "character")`: The get method for slot "colType"

colType<- `signature(object = "character")`: The set method for slot "colName"

dropOrNot `signature(object = "logical")`: The get method for slot "dropOrNot"

dropOrNot `signature(object = "logical")`: The set method for slot "dropOrNot"

Author(s)

Jianhua Zhang

See Also

`importWizard`

Examples

```
  newInfo <- new("colInfo", colName = "aaaa", colType = "character",
dropOrNot = FALSE)
```

`dataViewer` *Function to view a data object passed*

Description

This function creates a widget to allow users to view the content of a data frame passed and decide whether to save the data or not.

Usage

```
dataViewer(data, caption = "", save = TRUE)
```

Arguments

`data`	`data` a data frame (or alike) to be viewed
`caption`	`caption` a character string for the title of the widget
`save`	`save` a boolean to indicate whether to have the option to allow users to save the data

Details

Taking a data frame as one of the arguments, this function builds a widget that allows users to view the content of the data and save the data as a file.

Value

This function does not return any value

Author(s)

Jianhua Zhang

References

R tcltk

See Also

importWizard

Examples

```
# Create matrix
data <- matrix(1:20, ncol = 4)
if(interactive()){
    # View data using dataViewer
    dataViewer(data, "test", TRUE)
}
```

`dbArgsWidget` *Function to build a widget for inputing database arguments*

Description

This functions creates a interactive widget to allow users to input arguments for database connection for Unix.

Usage

```
dbArgsWidget()
```

Details

Database arguments include database name, user name, password, host name, and table name.

Value

The function returns a list containing the following elements:

dbname	a charater string for the name of the database
host	a character string for the name or IP address of the host machine
user	a character string for the name of the user
password	a character string for the password
tablename	a character string for the name of the database table

Author(s)

Jianhua Zhang

References

Rdbi

See Also

argsWidget

Examples

```
if(interactive()){
    test <- dbArgsWidget()
}
```

DPExplorer *Functions constuct a widget to explore BioC's data packages*

Description

These functions construct a widget that allow users to visually explore a data package of Bioconductor and read selected elements to R

Usage

```
DPExplorer(pkgName = "", title = "BioC Data Package Explorer", getFocus
= TRUE)
getTopLevel(title)
loadDataPkg(pkgName)
```

Arguments

pkgName	pkgName a character string for the name of a Bioconductor's data package that has already been loaded
title	title a character string for the title of the widget
getFocus	getFocus a boolean indicating whether a widget should grab the focus

Details

If pkgName is not provided when DPExplorer is called, an entry box is available for users to put a pkgName in later. In either cases, the data package specified by pkgName should have been loaded.

getTopLevel creates a top level window for the widget.

loadDataPkg filters out valid environment objects from a data package.

Value

If only one key is selected, DPExplorer returns a vector of one to more elements. If more than one key is selected, loadDataPkg returns a list of vectors.

getTopLevel returns a tkwin object for the top level window.

loadDataPkg returns a vector of character strings for available data environments.

Author(s)

Jianhua Zhang

References

Documents for a Bioconductor data package

Examples

```
if(interactive() && require("hgu95av2", character.only = TRUE)){
    DPExplorer("hgu95av2")
}
```

---

eExplorer *A widget that allows users to explore the example code and help files of a given R package*

Description

Given a valid package name as a character string, eExplorer collects all the example code from the "R-ex" directory from the R library for that package and then displays the names of the code examples in a list box. When a name in the list box is clicked, the corresponding code will be displayed and users are allowed to execute the code or view the help file for the function the example code is for.

Usage

```
eExplorer(pkgName, font = "arial 13", getFocus = TRUE)
getExCode(pkgName)
getHelpFile(pkgName, fileName)
```

Arguments

pkgName	pkgName a character string for the name of an R package of interest. The R package needs to be installed
font	font a character string for the font to be used by the widget to display the text. The default is "arial 13"
fileName	fileName a character string for the name of a file in "R-ex" with the ".R" extension removed. The file contains a chunk of example code and may have a corrsponding help file in the "help" directory
getFocus	getFocus a boolean indicating whether a widget should grab the focus

Details

getExCode and getHelpFile are called by eExplorer to get the code examples or help help files contained by a given package.

Value

eExplorer does not return anything useful.

Author(s)

Jianhua Zhang

References

Writing R Extension for information on "R-ex" and "help" directories

See Also

vExplorer

Examples

```
if(interactive()){
    require("Biobase") || stop("Does not run without Biobase")
    eExplorer("Biobase")
}
```

---

fileBrowser *Simple Interface to View and Pick Files*

Description

This function provides the widget for users to go up and down a path and view the files. When files are selected and the "End" button is pressed, the function returns a vector of character strings with the full paths of the selected files.

Usage

```
fileBrowser(path="", testFun = function(x) TRUE, prefix = NULL,
     suffix = NULL, textToShow = "Select file(s)",
     nSelect = -1)
```

Arguments

path	character string for the full path to be view. Defaults to the current working directory if no path is provided.
testFun	function that checks to see if a given file name satisfies predefined requirements. The default is no checking.
prefix	character string for a prefix used to screen out file names that do not have that prefix as part of their names.
suffix	character string for a suffix used to screen out file names that do not have that suffix as part of their names.
textToShow	character string to be shown on the widget to given instructions to users.
nSelect	integer indicating the number of files that should be selected. No limitation if nSelect = -1 as per default.

Details

When a path is viewed, files will be displayed as they are and directories will be displayed with a system file separator appended (e.g. "/" for Unix systems). Single click on a file name will make the file selectable when the select button is pressed. Multiple selection is possible by dragging with mouse button 1 pressed. Double click on a directory name will go into that directory and display its files. When a file/directory is selected and the "End" button pressed, the full path for the selected files selected will be returned.

The widget is modal and has to be closed by pressing the "End" button before doing any other operations. Functions, prefix, and suffix can be specified. Examples of validity functions are hasPrefix and hasSuffix.

The following is a list of the buttons and their associated behavior:

Up Moves the directory whose content is to be displayed in the box for file names one level up along the directory tree. No action if already on top of the tree.

Select \>\> When a file or files in the box for file names in a directory have been highlighted by clicking or dragging mouse button 1 and this button is pushed, the highlighted file(s) will be displayed in the box for selected file(s) on the right.

\<\< Remove When a file or files in the box for selected files have been highlighted by clicking or dragging this button is pushed, the highlighted file(s) will be removed from the box.

Clear Clears everything in the box for selected files when pushed.

end Returns a vector containing all the names in the box for selected files or NULL if the box is empty. The full path will be appended to the file names.

Value

A vector of character strings containing the full path of each file selected.

Author(s)

Jianhua (John) Zhang

See Also

pickFiles, hasPrefix, hasSuffix

Examples

```
## The example here is only run interactively since it requires user
## interference which may cause problems if not available:
if(interactive()) {

  # Call the function to view the current directory
  flist <- fileBrowser()
  flist

  # To call the function with a path do
  # fileBrowser(path = "yourPath")
}
```

---

fileWizard *A function that import a text file into R*

Description

Given a file name, this function imports the text file into R.

Usage

```
fileWizard(filename = "", fun = read.table, file = "file",
basic = c("header", "sep"), getFocus = TRUE)
```

Arguments

filename	A character string for the name of the text file to be imported
fun	An R function that is going to be used to read the file. Default to read.table
file	A character string for the name of the argument to fun that defines the name of the file to be read
basic	A vector of character strings for names of the arguments to fun that will have separate entry boxes on the widget to be produced. Default to "header" and "sep"
getFocus	getFocus a boolean indicating whether a widget should grab the focus

Details

This function is only partially finished and will be improved soon. It currently allows uesrs to view a given file and change the settings for header and sep arguments of read.table. A file will be read in based on the values of the two arguments and return.

Value

This function returns a data frame for the file read in.

Author(s)

Jianhua Zhang

References

R News Vol. 1/3, September 2001

See Also

fileBrowser

Examples

```
if(interactive()) {
  # Only the interface is displyed as no real file is given
  fileWizard()
}
```

---

getLightTW *Function to create a light weight widget showing a text string*

Description

Given a text string and coordinations, this function creates a light weight tcltk widget with showing the text string passed.

Usage

```
getLightTW(x, y, text)
```

Arguments

x	x an interger for the horizontal position for the widget to appear
y	y an integer for the vertical position for the widget to appear
text	text a character string to be show in the widget

Details

When the function is invoked, a box containing the text will appear at the position specified by x, and y. Click the widget makes it disappear.

Value

This function does not return any value

Author(s)

Jianhua Zhang

Examples

```
if(interactive()){
    getLightTW(200, 200, "Click Me!")
}
```

`getWvalues` *Obtaining values of widgets on a given widget*

Description

This function returns a list containing the values for widgets on a widget created by function widgetRender. It takes a list defining the widget and returns a named list containing the values for each of the widgets.

Usage

```
getWvalues(W)
```

Arguments

`W`	`W` a list of lists defining the widgets that are used to make a widget

Details

For a widget containing 3 widgets each with some associated functionalities, the list is defined as this:

pW1 <- list(Name="AAA", Value="bbb", toText=function(x) paste(x,collapse= ","), fromText=NULL, canEdit=TRUE, buttonFun = fileBrowser, buttonText = "Browse")

pW2 <- list(Name="BBB", Value="x,y,z", toText=function(x) paste(x, collapse=","), fromText=NULL, canEdit=TRUE, buttonFun = ls, buttonText = "List")

pW3 <- list(Name="CCC", Value="ccc", toText=function(x) paste(x, collapse = ","), fromText=NULL, canEdit=TRUE, buttonFun=NULL, buttonText=NULL)

widget1 <- list(wList = list(a = pW1, b = pW2, c = pW3), preFun = function() "Hi", postFun = function() "Bye")

widget1 will be used to create a widget with 3 entry boxes. When users modify the values through the widget created, new values will be kept in the list and widget1 will be returned up exist. getWvalues is useful to capture the values for each widgets on the widget.

Value

This function returns a list of:

comp1	Description of 'comp1'
comp2	Description of 'comp2'

Author(s)

Jianhua Zhang

See Also

widgetRender

Examples

```
# Define the widgets
  pW1 <- list(Name="AAA", Value="bbb",
            toText=function(x) paste(x,collapse= ","), fromText=NULL,
            canEdit=TRUE,
            buttonFun = fileBrowser, buttonText = "Browse")

  pW2 <- list(Name="BBB", Value="x,y,z",
            toText=function(x) paste(x, collapse=","), fromText=NULL,
            canEdit=TRUE, buttonFun = ls, buttonText = "List")

  pW3 <- list(Name="CCC", Value="ccc",
            toText=function(x) paste(x, collapse = ","), fromText=NULL,
            canEdit=TRUE, buttonFun=NULL,  buttonText=NULL)

  widget1 <- list(wList = list(a = pW1, b = pW2, c = pW3),
                preFun  = function() "Hi",
                postFun = function() "Bye")

  if(interactive()){
      tt <- widgetRender(widget1, "try")
      getWvalues(tt)
  }else{
      getWvalues(widget1)
  }
```

`guess.sep` *Automatically determines whether a data file has a header and what delimiter is used to separate data columns*

Description

This function reads a few lines from a data text file and determines whether a header exists, what the delimiter, and what data type each column is for a given file.

Usage

```
guess.sep(file.name, numLine = 5, seps = "", isFile = TRUE)
guess.header(twoLines, sep)
find.type(file.name, sep, header = FALSE, numLine = 5, isFile = TRUE)
charOrNum(vect)
getRowNames(file.name, sep, header, skip)
```

Arguments

`file.name`	`file.name` a character string for the name of the file of interests
`numLine`	`n` an integer or character indicating the total number of lines to be read from the file for making the determination
`seps`	`seps` a vector of characters of potential delimiters to be used to make the determination. Default delimiters include " ", ",", ";", and "\t". Anything inaddition to the default will have to be included in seps
`twoLines`	`twoLines` a vector of character string including the first two lines of a file that will be used to determine whether the file has a header
`sep`	`sep` a character for the delimiter used to separate columns in a file
`vect`	`vect` a vector of character or numeric string
`header`	`header` a boolean indicating whether a file has headers
`isFile`	`isFile` a boolean that is TRUE when `file.name` is a file or FALSE an object
`skip`	`skip` an integer for the number of lines to be skiped using `read.table`

Details

guess.sep calls `guess.sep` and `find.type` to determine the header, delimiter, and column data type of a file.

`charOrNum` determines which elements of a vector are numeric or character.

Value

This function returns a list containing

`header`	TRUE if there is a header and FALSE otherwise
`separater`	A character string indicating the delimiter used
`type`	A vector of character strings that are either character or numeric

Author(s)

Jianhua Zhang

See Also

fileWizard

Examples

```
# Create a temp file
tempData <- matrix(1:20, ncol = 4)
write.table(tempData, file = "tempData", quote = FALSE, sep =
"\t", row.names = FALSE, col.names = TRUE)

guess.sep("tempData")

unlink("tempData")
```

---

hasChar *String Prefix and Suffix Checking*

Description

These functions return a *function* for determining if a given prefix, suffix, or set of characters passed to this function exists in a character string passed to the returned function.

Usage

```
hasChar(toCheck, what = "")
hasPrefix(aPrefix)
hasSuffix(aSuffix)
```

Arguments

aPrefix	character string to be used as the prefix to be checked for
aSuffix	character string to be used as the suffix to be checked for
toCheck	toCheck a character string to be used to check to see if it exists in a character string passed to the returned function
what	what a character string defining whether toCheck will be used as a prefix (what = "prefix"), suffix (what = "suffix"), or a set of characters (what = "") to check the character string passed to the returned function

Details

The prefix (or suffix) is passed to hasPrefix (or hasSuffix) and then the returned function can be used to check for the existence of that prefix (suffix) in a string passed to that function.

hasChar is a more general function that determines the existence of prefix, sufix, or a set of a characters in a character string passed to the returned function.

Value

A **function** which itself returns a logical (of length 1) indicating if the prefix/suffix is found (TRUE) or not.

Author(s)

Jianhua (John) Zhang

See Also

pickFiles

Examples

```
# Function for checking for a prefix "xxx" :
chkPfun <- hasChar("xxx", what = "prefix")
# Function for checking for a suffix ".tex" :
chkSfun <- hasChar(".tex", what = "suffix")

chkPfun("xxx.tex")
chkPfun(".xxx")
chkSfun("xxx.tex")
chkSfun("yyyyy")
chkSfun("yxxx.text")
```

---

importPhenoData *Functions to input data for an AnnotatedDataFrame object*

Description

This functions allow users to read data from an existing file or an R data.frame object and use the data frame to construct an AnnotatedDataFrame object.

Usage

```
importPhenoData(fileName, sampleNames = NULL, from = NULL)
createPData(pdata, varList)
writePDRowNames(pdata, sampleNames)
writePhenoTable(base, textWidget, pdata)
makePhenoData(pdata)
convert2PData(phenoList)
getOBJWidget(type = NULL)
objExists(name, type = NULL)
getSNCNums(sampleNames)
getCovarDesc(varList)
```

Arguments

fileName	a character string for the name of a file that is going to be used to build an AnnotatedDataFrame object.
sampleNames	a vector of character strings for the names of samples. The length of sampleNames should be the same as the number of rows of an existing file or data.frame if an AnnotatedDataFrame object is to be created based on a file or data.frame.
pdata	a data.frame for the experimental data.
base	an RTcl object for the base window a widget resides.
textWidget	an RTcl object for a text box widget.
phenoList	a list of lists for tclVar() objects.
type	a character string for the class of a object e.g. data.frame, AnnotatedDataFrame.
name	a character string for the name of an object.
varList	a list of characters with names being covariate names and values being short descriptions of covariate names.
from	a character string indicating how an AnnotatedDataFrame object will be created. "file" - create from an existing file, "object" - create from an existing data frame object, "edit" - create by editing an existing AnnotatedDataFrame object, and "new" create a new AnnotatedDataFrame object from scratch. NULL or any other values for from will invoke a widget that allows users to select one of the four means from an interface.

Details

When import a data.frame or AnnotatedDataFrame object, the object to be imported should have been stored in .GlobelEnv. All the objects of data.frame or AnnotatedDataFrame will be made available through a browser.

The main widget if importPhenoData that calls other functions/widgets to have the job done.

Package Biobase is required for importPhenoData but the requirement id not forced as it is the only time the package is used. Users have to make sure that Biobase is available.

Value

An AnnotatedDataFrame object.

Note

This function is intended for use by function read.phenoData of Biobase

Author(s)

Jianhua Zhang

References

AnnotatedDataFrame class in Biobase

See Also

AnnotatedDataFrame-class

Examples

```
if(interactive()){
    importPhenoData()
}
```

---

importWizard *A widget for importing data to R*

Description

Functions constructs a widget that allows users to inport data file to R. The imported data will be returned as an R data frame together with the argument list used to import the data using read.table

Usage

```
importWizard(filename = "", maxRow = 400)
initImportWizard(env)
getTopCan(base, env)
getAFrame(base, env)
finish(env)
getState1Frame(base, env)
setState1BFrame(frame, env)
setState1TFrame(frame, viewer, delims, env, startList)
showData4State1(widget, env)
setState1MFrame(frame, env, dataViewer)
getState2Frame(base, env, state = "state2", reset = FALSE)
setState2MFrame(frame,env)
setSepRadios(frame, env, state = "state2")
setQuoteList(frame, env)
setQuote(listBox, env, state = "state2")
setState2BFrame(frame, env)
showData4State2(canvas, env, state = "state2")
getState3Frame(base, env)
setState3TFrame(frame, env)
setState3BFrame(frame, env)
getName4Data(filename, objType)
writeCol4Matrix(tempFrame, dataFile, colInfos, env)
popStartLine(startList, env)
readFileByLines(filename)
```

Arguments

filename	filename a character string for the name of the file to be imported. The default is an empty string and users have to click a browse button to get the file name through fileBrowser
maxRow	maxRow an integer for the maximum number of rows of the data file to be imported to be shown on the widget. The default is 200 rows
env	env an R environment object for storing the required information
base	base a tcltk window to hold a canvas on the top and frames in the bottom
frame	frame a tcktl frame
viewer	viewer a tkwin object for a widget
delims	delims a character string for a file separater
widget	widget a tcltk widget
state	state a character string for the state of importing process
listBox	listBox a tcltk list box
canvas	canvas a tcltk canvas
tempFrame	tempFrame a tcltk frame that will be used to hold widget elements
dataFile	dataFile a data matrix holding data to be displayed
colInfos	colInfos an object of class colInfo with a name, type, and drop slot
reset	reset a boolean that is TRUE when the window needs to be reset
dataViewer	dataViewer a tkwin object for a list box
objType	objType a character string indicating the data type of an object to be saved. Defaulted to "object"
startList	startList a tk text box object

Details

importWizard mimics the interface of MS Excel and collects arguments for the function read.table. Due to performace concern, a maximum number of rows (maxRow) set by users will be displayed. Overly long data set may cause slow response of the system.

initImportWizard initializes the interface for importWizard by creating a widget with an empty top canvas and bottom frame filled with four buttons.

getTopCan Creates a canvas that is going to be filled by a frame created by other functions depending on the state of the importing process.

getAFrame Gets a frame for the canvas created by initImportWizard based on the current state of importing process.

finish Finishes the importing process and returns a data frame read from a file using read.table.

getState1Frame Returns a tcltk frame containing a list box to show a data file read by readLines and widgets for user imports.

setState1BFrame Fills the bottom frame of the frame created by getState1Frame with a list box.

setState1TFrame Fills the top frame of the frame created by getState1Frame with a list box.

showData4State1 Populates a tcltk list or text widget with data read using readLines.

setState1MFrame Fills the mid frame of the frame created by getState1Frame.

getState2Frame Returns a tcltk frame containing a canvas to show a data file read by read.table and widgets for user imports.

setState2MFrame Fills the mid frame of the frame created by getState2Frame.

setSepRadios Renders radio buttons for options of file separators in the frame created by setState2MFrame.

setQuoteList Renders the selection list for the quote used by a data file in the frame created by setState2MFrame.

setQuote Sets the value when a user has selected the quote used by a data file.

setState2BFrame Fills the bottom frame of the frame created by getState2Frame with a canvas.

showData4State2 Populates the canvas created by setState2BFrame using data read by read.table.

getState3Frame Returns a tcltk frame containing a canvas to show a data file read by read.table and widgets for user imports.

setState3TFrame Fills the top frame of the frame created by getState3Frame.

setState3BFrame Fills the bottom frame of the frame created by getState3Frame.

getName4Data Takes user input for a file name using a widget.

writeCol4Matrix Creates a tcltk frame with list boxes as columns displaying data of a data matrix.

Value

getTopCan returns a tcltk canvas.

getAFrame returns a tcltk frame.

finish returns a data.frame.

getState1Frame returns a tcltk frame with several widgets.

setState1BFrame returns the tkwin object of list box.

getState2Frame returns a tcltk frame with several widgets.

getState3Frame returns a tcltk frame with several widgets.

getName4Data returns a character string for the name of a file to be saved.

Author(s)

Jianhua Zhang

See Also

fileBrowser, argsWidget

Examples

```
if(interactive()){
    importWizard()
}
```

listSelect *Utilities Creating a Widget With Selection Boxes*

Description

These functions create a widget with selection boxes allowing users to view and make selections of items shown on the interface.

Usage

```
listSelect(aList, topLbl = "Select Elements From The Following List",
           typeFun = stdType, valueFun = stdView)
writeSelBox(baseW, aList, typeFun = NULL, valueFun = NULL)
   writeBut(baseW, butList, butWidth = 6)
writeLabel(baseW, typeFun, valueFun)
```

Arguments

aList	list with names and object pairs (e. g. a = "AAA").
topLbl	character string for the text to be shown as a title.
typeFun	function that takes an R object as an arguement and returns a description of the object.
valueFun	function that takes an R object as an argument and shows the content of the object. The function should get the representation of the object and calls objViewer to have the representation rendered in a widget.
baseW	a window widget to which the selection boxes will be put.
butList	a list with names and function pairs that define the name and behavior of buttons to be put on the widget to be generated
butWidth	numerical value specifying the width of buttons to be created.

Details

Both typeFun and valueFun have to take an argument (the R object to be shown). It works well for the valueFun function to call objViewer() with whatever to be shown passed to objViewer as an argument.

Value

listSelect() returns a list with the names of the R objects in the original list associated with TRUE (selected) or FALSE (deselected).

Author(s)

Jianhua Zhang

See Also

objViewer

Examples

```
aList <- list(a = "AAA", b = c(123, 456, 789),
              c = as.data.frame(matrix(1:10, ncol = 2)),
              d = stdType)
# Since user interference is required, the example code does not run
# automatically
if(interactive())
  listSelect(aList)
```

---

objectBrowser *View the Objects in the Workspace*

Description

This widget allows uers to view and select objects from the workspace. When the End button is pressed, the selected objects will be returned as a list.

Usage

```
objectBrowser(env = .GlobalEnv,fun = noAuto, textToShow = "Select object(s)", nSelect = -1)
```

Arguments

fun	function to test whether certain conditions are met by the objects. Only objects that meet the conditions will be displayed.
textToShow	character with the message to be shown on the widget as an instruction.
nSelect	integer indicating the number of objects to select. No limitation if nSelect = -1 as per default.
env	env a default environment object to start object Browser

Details

This function will return a list of lists with a "name" and "obj" pair for each object selected. The "name" will be the name of the object and "obj" will be the value of the object. If the object is a package, a description of the contents of the package will be the value. If the selected object is a function, a text string of the original code will be the value. A function can be passed to impose a filtering mechanisms on the objects to be displayed. See function isCharacter for an example of writing a filtering function for objectBrowser.

The buttons and their expected behavior are

Up Moves one level up along the search list and displays the content in the box for object names on the left of the widget.

`Select \>\>` When objects in the box for object names have been highlighted by clicking or clicking/dragging, this button will display the highlighted object names in the box for selected objects on the right.

`Reset` Moves back to `.GlobalEnv` which is the default starting point of the system.

`\<\< Remove` When object names in the box for selected objects have been highlighted by clicking or clicking/dragging and this button is pressed, the highlighted object names will be removed from the display.

`Clear` Removes all the object names from the box for selected objects.

`Cancel` Exits the widget and returns `NULL` when pressed.

`End` Returns a `list` of lists with names of the objects in the box for selected objects and their corresponding values or `NULL` if nothing exists in the box.

Value

A `list` of lists with a name and value pair for each object.

Author(s)

Jianhua (John) Zhang

Examples

```
## The example here is only run interactively since it requires user
## interference which may cause problems if not available:
if(interactive()) {

# Call the function with the isCharacter function.
r <- objectBrowser()
str(r) # show what we've got
}
```

---

objNameToList *Convert Object Names to List of Lists with (name, object) Pairs*

Description

This function supports `objectBrowser` by converting a vector of selected object names to a list of lists with object names and the corresponding objects.

Usage

```
objNameToList(objNames, env)
```

Arguments

`objNames`	character vector giving the names of objects.
`env`	an R environment where R objects are stored

Details

Each list in the list that is going to be returned contains a name for the object and the real value of the object. If the object name is a package name, the contents of the package will be the value associated with the package name.

Value

A list of lists each with a `name` and an `obj` component.

Author(s)

Jianhua (John) Zhang

See Also

`objectBrowser`

Examples

```
# Create two R objects
obj1 <- c("aaa", "bbb", "ccc")
env1 <- new.env(parent = baseenv())

# Get a list containing the two objects
nl <- objNameToList(c("obj1", "env1"), parent.frame())
str(nl)
```

---

objViewer *Show the Content of an R Object in a Widget*

Description

This function takes an R object and shows the content in a list box on a widget.

Usage

```
objViewer(toView, width = 40, height = 10)
```

Arguments

`toView`	R object whose content is to be viewed
`width, height`	positive values specifying the width and height of the widget.

Details

The function makes no check of the R object passed and will show whatever the object will be shown when the name is type at an R prompt. Formatting is required before passing the R object to the function.

Value

This function does not return any value

Author(s)

Jianhua Zhang

See Also

listSelect

Examples

```
# Since user interference is required, the example code only runs
# interactively
if(interactive())
   objViewer("Just to show that the content gets posted")
```

---

pExplorer *A widget to explore R packages*

Description

This widget allows users to explore R packages in the R library and try the example code.

Usage

```
pExplorer(pkgName = "", pkgPath = "", exclude = getExclude(), getFocus =
TRUE)
getPkgContents(pkgName, exclude = getExclude())
getFileContents(path, fileName)
getExclude()
getRPkgs(pkgPath)
hasDesc(pkgPath)
procRda(fileName)
procHelp(fileName)
procPDF(fileName)
procHTML(fileName)
```

Arguments

pkgPath	pkgPath a character string for the path where R packages are loacted
path	path a character string for the path of a given file
pkgName	pkgName a character string for the name (including path) of an R package to be explored
fileName	fileName a character string for the name (including path) of a file of interest

exclude	exclude a vector of character strings containing the directory or file names that will not be available for explorering. Package names have to have a system file separator appanded to the end (e. g. "/" under Unix)
getFocus	getFocus a boolean indicating whether a widget should grab the focus

Details

With or without a package name, the widget will have all the installed R package names in a drop-down list for user to select. As the default, the first element from list.files will be selected and the contents displayed if no package name is given.

getPkgContents gets the contents of a given R package and getFileContents gets the contents of a givan file.

getRPkgs, hasDesc, procRda, procHelp, procPDF, and procHTML are functions called by pExplorer to process different file or directory types.

Value

The widget returns invisiable()

Author(s)

Jianhua Zhang

References

Documentation on R packages

See Also

eExplorer

Examples

```
require("tkWidgets") || stop("tkWidgets not available")
getPkgContents(.libPaths(), "tkWidgets")
getFileContents(file.path(path.package("tkWidgets"), "help"),
    list.files(file.path(path.package("tkWidgets"), "help"))[1])
if(interactive()){
    pExplorer()
}
```

pickFiles *Pick Elements From Vector of Strings*

Description

Takes a vector of strings and then checks to see if the predefined conditions are met for each element. Elements that meet the conditions will be included in the vector returned and the others not.

Usage

```
pickFiles(fileNames, fun = function(x) TRUE,
          prefix = NULL, suffix = NULL, exclude = .Platform$file.sep)
```

Arguments

fileNames	vector of strings that will be checked.
fun	function to be used to check the strings. Default is no checking.
prefix	character used to check to see if strings in the vector have the prefix.
suffix	character used to check to see if strings in the vector have the suffix.
exclude	character string with which strings in the vector will be excluded form the checking. The default is to exclude all the directory names and always return them.

Details

The function fun will be used only when both prefix and suffix are NULL. If a prefix is not NULL, that prefix will be checked. A suffix is going to be checked when prefix is NULL.

Value

Character vector of file names satisfying the conditions.

Author(s)

Jianhua Zhang

See Also

fileBrowser, hasPrefix, hasSuffix

Examples

```
# Return every thing from the current directory
pickFiles(list.files())

# Create a temp file
file.create("myFile")
```

```
# Returns subdirectory names and file names with a prefix of "my"
pickFiles(list.files(), prefix = "my")

# create another temp file
file.create("temp.tex")

# Return subdirectory names and file names with a suffix of ".tex"
pickFiles(list.files(), suffix = ".tex")

# clearn up
unlink("myFile")
unlink("temp.tex")
```

`pickItems` *Function that builds a widget to allow users to select items from available sources*

Description

Given a vector of characters, this function creates a widget containing list box to allow users to visually select elements from the vector.

Usage

```
pickItems(items, title1 = "Items to pick", title2 = "Picked items")
```

Arguments

`items`	`items` a vector for the available source elements to be selected
`title1`	`title1` a character string for the title of the list box that shows the list of items to be selected from
`title2`	`title2` a character string for the title of the list box that shows the items that have been selected

Details

This function is to provide visual support to other functions and thus may not have much use otherwise.

Value

This function returns a vector of select items.

Author(s)

Jianhua Zhang

References

R tcltk

See Also

dataViewer

Examples

```
options <- paste("Option", 1:10, sep = "")
if(interactive()){
    pickItems(options)
}
```

---

pickObjs *Determine What to Be Sent to a Widget*

Description

This function takes a vector of object names and determines what will be sent to (e.g. the objectBrowser) widget for display based on the default and user input requirements.

Usage

```
pickObjs(objNames, fun = noAuto)
noAuto(x)
```

Arguments

objNames	objNames character vector with object names to be processed
fun	fun function checking the object names for satisfaction of certain requirement
x	x a character string for the name of an object

Details

Packages and environments are always displayed.

Value

Character vector of object names that satisfy the requirements.

Author(s)

Jianhua (John) Zhang

See Also

objectBrowser

Examples

```
# Returns names of package and environment objects in the search path.
pickObjs(search())
```

---

setArgsList *Functions to support importWizard*

Description

The functions are to support importWizard and may not have much practical use otherwise.

Usage

```
setArgsList(filename, env, isFile = TRUE, init = TRUE)
whatDeli(delimiter)
getMoreArgs()
assignArgs(value, env)
getArgs(env)
assignShowNum(value, env)
getShowNum(env)
assignCState(value, env)
getCState(env)
assignColInfo(value, env)
getColInfo(env)
setColInfos(types, env)
changeState(canvas, backBut, nextBut, env, forward = TRUE, endBut, viewBut)
setNewState(env, backBut, nextBut, forward = TRUE, endBut, viewBut)
addArgs(env)
dropArgs(env)
setSkip(widget, env, state = "state1")
moreArgs(env)
dropColumn(index, env)
setColName(index, entryBox, env)
setColType(index, entryBox, env)
assignLineData(lineData, env)
getLineData(env)
```

Arguments

filename	filename a character string for the full name of a file
env	env an R environment object for storing information
delimiter	delimiter a character string for the delimiter whose letter representation is sought
value	value a character or numerical value to be assigned to a variable
backBut	backBut a tkwin object for the button that shifts back to the previous state

nextBut	nextBut a tkwin object for the button that shifts to the next state
forward	forward a boolean indicating the direction of state change
widget	widget a tcltl widget
state	state a character string for the state of importing process
index	index an integer for the index of the list for column information
entryBox	entryBox

a tcltk entry box.

canvas	canvas a tcltk canvas
types	types a vecter of string indicating the types of data columns
lineData	lineData a vector of character strings read in using readLines
endBut	endBut a tkwin object for the button that ends the process when pressed
viewBut	viewBut a tkwin object for the button that refresh the window when pressed
init	init a boolean that is TRUE when the widget is first set up and FALSE otherwise
isFile	isFile a boolean that is TRUE if fileName is a file

Details

setArgsList calls function guess.sep to figure out the the header, sep, and data type of a file and sets the values for argument list and colInfo.

whatDeli gets the word representation of delimiters (e.g. tab for "\t").

getMoreArgs generates a widget using widgetTools to collect some of the arguments for read.table.

assignArgs updates "argsList" stored in a predefined environment.

getArgs Gets "argsList" from a predefined environment.

assignShowNum Updates the value for "showNum" (number of rows to show in the interface.

getShowNum Gets the value for "showNum" (number of rows to show in the interface.

link{assignCState} Updates the value of "currentState" that is stored in a predefined environment.

getCState Gets the vlaue of "currentState" that is stored in a predefined environment.

assignColInfo Updates the values of "colInfos" (column information) that is stroed in a predefined environment.

getColInfo Gets the values of "colInfos" (column information) that is stroed in a predefined environment.

setColInfos Creates colInfo objects and sets the value of 'colInfos' list.

changeState changes the state and thus the interface of a widget.

setNewState sets the state of a importing process.

addArgs adds a new state to the argument list for states.

dropArgs removes a state from the argument list for states.

setSkip Sets the value for the number of lines to skip when readling a data file.

`moreArgs` Gets some of the arguments for importing data using `read.table`.

`dropColumn` Sets the index values for data columns that are going to be droped when read using `read.table`.

`setColName` Sets the column names for a data file by getting column names from correct entry boxes.

`setColType` Sets the column type for a data file by getting column type information from correct entry boxes.

Value

`whatDeli` returns a character string.

`getMoreArgs` returns a list of arguments.

`getArgs` returns a list of the arguments for read.table.

`getShowNum` returns an integer for the number of rows to show.

`getCState` returns a character string for the current state.

`getColInfo` returns a colInfo object contains column information.

Author(s)

Jianhua Zhang

See Also

`importWizard`

Examples

```
# No example is given as functions require the set up of the working
# environment.
```

`stdType` *Provide Default Behavior for listSelect Helper Functions*

Description

The function `listSelect` takes two functions which define how the type information and content of R objects will be shown on the widget created by `listSelect`. Functions `stdType()` and `stdView()` provide the default behavior.

Usage

```
stdType(toCheck)
stdView(toView)
```

Arguments

toCheck, toView arbitrary R object.

Details

These functions can be viewed as exmaples of defining functions for the typeFun and valueFun arguments of the listSelect function.

Value

stdType() returns a character string describing the type of the R object.

stdView()

Author(s)

Jianhua Zhang

See Also

listSelect

Examples

```
stdType(123)
stdType("What am I")

str(mydf <- data.frame(x = 2:8, ch = letters[1:7]))
stdType(mydf)# "list"
stdType(stdType)

if(interactive()) {## stdView() needs UI:
  stdView(1:10)
  stdView(mydf)
 }
```

---

tkMIAME *Simple Interface to enter experimental design information*

Description

This function provides a widget for users to enter experimental design MIAME information.

Usage

```
tkMIAME()
```

Details

This widget provides an interface to enter experimental information following the MIAME (Minimum Information About a Microarray Experiment) standard.

A draft of the latest document (v. 1.1) is http://www.mged.org/Workgroups/MIAME/miame_1.1.html

Brazma et al. divide the MIAME into 6 sections 1. Experimental design, 2. Array design, 3. Samples, 4. Hybridizations, 5. Measurements, and 6. Normalization controls This widget is for the first section. We ask for the user to enter: experimenter name, laboratory, contact information, a single-sentence experiment title, an abstract describing the experiment, URLs. This slot could also include a formal statistical description of the experimental design (e.g. using factors). Some of this info is already stored in AnnotatedDataFrame or elsewhere.

The function returns a list that is intended for the creation of an object of class MIAME. However, we return a list so that the function can work independently of the Biobase package.

Value

A list containing entries:

ExperimentName	character string
LabName	character string
ContactInfo	character string
ExperimentTitle	character string
Description	character string
URL	character string

Author(s)

Majnu John

References

"Minimum information about a microarray experiment (MIAME)-toward standards for microarray data", A. Brazma, et al., Nature Genetics, vol. 29 (December 2001), pp. 365-371, http://www.mged.org/Workgroups/MIAME/miame_1.1.html

See Also

MIAME

tkphenoData *Simple interface to enter AnnotatedDataFrame*

Description

This widget provides an interface to create AnnotatedDataFrame-class instances.

Usage

```
tkphenoData(sampleNames)
```

Arguments

sampleNames	sampleNames for which we will enter phenotypic data.

Details

The function returns a list of character matrices intended to be used as the pData and varLabels slots of an instance of AnnotatedDataFrame-class.

Value

A list of two matrices

pData	a character matrix containing phenotypic data.
varLabels	a character vector with covariate description.

Author(s)

Majnu John

See Also

AnnotatedDataFrame-class

tkSampleNames *Simple interface to associate sample names with files*

Description

This widget provides an interface to enter names to be associated with files containing array expression information related to a particular sample.

Usage

```
tkSampleNames(..., filenames = character(0))
```

Arguments

...	the filenames to be associated with a sample name, supplied individually and/or as a character vector
filenames	a character vector of filenames to be associated with a sample name.

Details

AnnotatedDataFrame-class objects will use sample names as row names for its pData. The colnames of the expression matrices in ExpressionSet-class use this as well. Many times, each of these columns are obtained from a file. Rather than use the, sometimes ugly, filename we can use the sample names that this interface associates with each file.

The function returns a character matrix intended to be used to create sample names in AnnotatedDataFrame and ExpressionSet. However, the function can be used independently of the Biobase package.

Value

A character matrix with the first column the filenames the second column the sample names to associate.

Author(s)

Majnu John

See Also

AnnotatedDataFrame-class

---

values.Widget *Deal with Names and Values of Widget Created by widgetRender()*

Description

Functions in this group print or list the names or/and values of the widget elements on a widget created by widgetRender().

Usage

```
values.Widget(x)
```

Arguments

x	A list (print.pWidget) or list of lists(print.Widget, values.Widget) that representing a widget element (list) on a widget or a widget (list of lists) generated by using the function widgetRender.

Details

print.pWidget takes a list defining a widget element on a widget generated by using the function widgetREnder. An example of a valid list will be:

pW1 <- list(Name="AAA", Value="bbb", toText=function(x) paste(x,collapse = ","), fromText=NULL, canEdit=TRUE, buttonFun = fileBrowser, buttonText = "Browse")

print.Widget and values.Widget take a list of lists defining all the widget elements on a widget generated using the function widgetRender. An example of a valid list will be:

pW1 <- list(Name="AAA", Value="bbb", toText=function(x) paste(x,collapse = ","), fromText=NULL, canEdit=TRUE, buttonFun = fileBrowser, buttonText = "Browse")

pW2 <- list(Name="BBB", Value="x,y,z", toText=function(x) paste(x, sep=","), fromText=NULL, canEdit=TRUE, buttonFun = ls, buttonText = "List")

pW3 <- list(Name="CCC", Value="ccc", toText=function(x) paste(x, collapse = ","), fromText=NULL, canEdit=TRUE, buttonFun=NULL, buttonText=NULL)

widget1 <- list(wList = list(a = pW1, b = pW2, c = pW3), preFun = function() "Hi", postFun = function() "Bye")

Value

`returnList` values.Widget returns a list of lists each with the name and value of an entry box on the widget created.

Author(s)

Jianhua (John) Zhang

See Also

widgetRender

Examples

```
# Create the lists and list of lists
  pW1 <- list(Name="AAA", Value="bbb",
              toText=function(x) paste(x,collapse = ","),
              fromText=NULL, canEdit=TRUE, buttonFun = fileBrowser,
              buttonText = "Browse")

  pW2 <- list(Name="BBB", Value="x,y,z", toText=function(x) paste(x, sep=","),
            fromText=NULL, canEdit=TRUE, buttonFun = ls,
           buttonText = "List")

  pW3 <- list(Name="CCC", Value="ccc",
              toText=function(x) paste(x, collapse = ","),
              fromText=NULL, canEdit=TRUE, buttonFun=NULL,
              buttonText=NULL)

  widget1 <- list(wList = list(a = pW1, b = pW2, c = pW3),
                  preFun = function() "Hi",
```

```
                  postFun = function() "Bye")

# Define the classes
class(pW1) <- c("pWidget", "textbox")
class(widget1) <- "Widget"

# Call the funcitons
print.pWidget(pW1)
print.Widget(widget1)
values.Widget(widget1)
```

---

vExplorer *An interface to interact with vignette code chunks*

Description

This function provides a widget for viewing, editing, and executing code chunks of vignettes.

Usage

```
vExplorer(title = "BioC Vignettes Explorer", pkgName = "", font =
ifelse(.Platform$OS.type == "unix", "arial 14", "arial 11"))
viewVignette(title, packName, vigPath, font = "arial 11")
```

Arguments

title	character string for the name to be displayed as the title of the widget to interact with code chunks.
pkgName	vector (of length 1 for pkgName) of character strings for names of Bioconductor packages the code chunks of whose vignettes will be explored.
packName	same as pkgName
vigPath	character string for the full qualified name of a vignette to be explored.
font	a character string for the name and size of the font to be used for text rendered on the widgets (e. g. "arial 11")

Details

By default, packNames = "", all the installed packages will be examined and those that have vignettes will be listed to allow users to choose from.

Value

This function does not return any useful value.

Note

This function is part of the Bioconductor project at Dana-Faber Cancer Institute to provide Bioinformatics functionalities through R.

Author(s)

Jianhua Zhang

References

http://www.bioconductor.org

Examples

```
if(interactive()){
    require("DynDoc", character.only = TRUE)
    require("tools", character.only = TRUE)
    require("widgetTools", character.only = TRUE)
    vExplorer()
    path <- path.package("widgetTools")
    vigList <- pkgVignettes("widgetTools")
    viewVignette("BioC VignetteBrowser", "widgetTools", vigList$docs)
}
```

---

widgetRender *Render a Tk Widget from Given Specifications*

Description

This function takes a list that specifies the appearance and behavior of a Tk widget and renders the widget accordingly.

Usage

```
widgetRender(iWidget, tkTitle)
```

Arguments

iWidget	list of lists that specifies the appearance and behavior of the widget to be rendered.
tkTitle	character string for the text to appear in the title bar of the widget to be rendered.

Details

The widget to be rendered normally consists of frames with three widgets arranged in a row. The first widget is normally a label for the name of the second widget. The second widget can be any type of widgets. The third widget is a button widget that defines some behavior to be associated with the second widget. For example, a button that will cause something to be displayed in the second widget when pressed. The third widget can be missing if no such association is required.

The widget to be rendered also has two buttons at the bottom part of the widgets. The followings are the name and behavior of the buttons:

Cancel - The unmodified list passed to the function at the time of invocation will be returned when pressed.

End - A modified version of the `iWidget` argument will be returned when pressed. The returned list has the same number of elements as the original one but with the values modified based on the entries in corresponding widgets items.

Value

A `list` of lists with the original values of the passed modified or unmodified depending on whether the cancel or end button pressed.

Author(s)

Jianhua (John) Zhang

References

Peter Dalgaard (2001) A Primer on the R-Tcl/Tk Package; R News **1** (3), 27–31 http://CRAN.R-project.org/doc/Rnews/

See Also

`fileBrowser`, `objectBrowser`.

Examples

```
# Create the list to be passed
pW1 <- list(Name="AAA", Value="bbb",
            toText=function(x) paste(x,collapse= ","), fromText=NULL,
            canEdit=TRUE,
            buttonFun = fileBrowser, buttonText = "Browse")

pW2 <- list(Name="BBB", Value="x,y,z",
            toText=function(x) paste(x, collapse=","), fromText=NULL,
            canEdit=TRUE, buttonFun = ls, buttonText = "List")

pW3 <- list(Name="CCC", Value="ccc",
            toText=function(x) paste(x, collapse = ","), fromText=NULL,
            canEdit=TRUE, buttonFun=NULL,  buttonText=NULL)

widget1 <- list(wList = list(a = pW1, b = pW2, c = pW3),
                preFun  = function() "Hi",
                postFun = function() "Bye")

# Call the function
if(interactive()){
    x <- widgetRender(widget1, "Test Widget")
    str(x)
}
```

WName *Accessors for Primitive Widget Objects*

Description

Currently primitive widget objects (pwidgets) are implemented as lists, but this will change. Users should only rely on these accessors functions, not the implementation.

Usage

```
WName(x)
WValue(x)
WValue(x) <-  value
WtoText(x)
WfromText(x)
WcanEdit(x)
WbuttonText(x)
WbuttonFun(x)
WwList(x)
WwList(x) <- value
WLValue(x, index)
WLValue(x, index) <- value
WRButtons(x)
WpreFun(x)
WpostFun(x)
WEnd(x)
```

Arguments

x	list of lists with a Name, Value, toText, fromText, canEdit, buttonText, buttonFun, preFun, postFun element.
index	integer or character string as an index or name for the list in the first list of a list of lists for a widget.
value	An R data type that is going to be used to update the value.

Details

`WName(x)` returns the `Name` element of `x`. `WValue(x)` returns the `Value` element of `x`. `WValue(x) <- value` will change the `Value` element of `x` to `value`.

`WtoText(x)` returns the `toText` element of `x`. `WfromText(x)` returns the `fromText` element of `x`. `WcanEdit(x)` returns the `canEdit` element of `x`.

`WbuttonText(x)` returns the `buttonText` element of `x`. `WbuttonFun(x)` returns the `buttonFun` element of `x`. `WwList(x)` returns the `wList` element of `x`. `WwList <- value` will update the `wList` element of a Widget list with `value`.

`WLValue(x, index)` returns a list indicated by the index in the wList of a widget list. `WLValue(x, index) <- value` will update indicated by the index in the wList of a widget list with the value.

`WpreFun(x)` returns the preFun element of `x`. `WpostFun(x)` returns the post element of `x`. `WEnd(x)` returns the end element of `x`.

Value

A character string or R function represented by the element of the list whose value is to be retrieved.

Author(s)

Jianhua (John) Zhang

See Also

`widgetRender` for references etc.

Examples

```
# Create the list of lists
pW1 <- list(Name="AAA", Value="bbb",
            toText = function(x) paste(x,collapse = ","),
            fromText = NULL, canEdit = TRUE,
            buttonFun = ls, buttonText = "Browse")

widget1 <- list(wList = list(a = pW1),
                preFun = function() "Hi",
                postFun = function() "Bye")

# Call the functions
WName(pW1)
WValue(pW1)
WValue(pW1) <- "lll"
WtoText(pW1)
WfromText(pW1)
WcanEdit(pW1)
WbuttonText(pW1)
WbuttonFun(pW1)
WwList(widget1)
WwList(widget1) <- list(Name = "New list", Value = "New value")
WLValue(widget1, 1)
WLValue(widget1, 1) <- "New value"
WpreFun(widget1)
WpostFun(widget1)
WEnd(widget1)
```

Index

∗ **classes**
colInfo-class, 5
∗ **interface**
argsWidget, 4
dataViewer, 6
dbArgsWidget, 7
DPExplorer, 8
eExplorer, 9
fileBrowser, 10
fileWizard, 12
getLightTW, 13
importPhenoData, 18
importWizard, 20
listSelect, 23
objectBrowser, 24
objViewer, 26
pExplorer, 27
pickItems, 30
tkMIAME, 35
tkphenoData, 37
tkSampleNames, 37
vExplorer, 40
widgetRender, 41
∗ **manip**
appendSepDir, 2
hasChar, 17
objNameToList, 25
pickFiles, 29
pickObjs, 31
values.Widget, 38
WName, 43
∗ **misc**
args2XML, 3
getWvalues, 14
guess.sep, 16
setArgsList, 32
stdType, 34
.GlobalEnv, *25*

addArgs, *33*
addArgs *(*setArgsList*)*, 32
AnnotatedDataFrame, *36*
appendSepDir, 2
args2XML, 3
argsWidget, 4, *5*, *8*, *22*
assignArgs, *33*
assignArgs *(*setArgsList*)*, 32
assignColInfo, *33*
assignColInfo *(*setArgsList*)*, 32
assignCState *(*setArgsList*)*, 32
assignLineData *(*setArgsList*)*, 32
assignShowNum, *33*
assignShowNum *(*setArgsList*)*, 32

changeState, *33*
changeState *(*setArgsList*)*, 32
charOrNum, *16*
charOrNum *(*guess.sep*)*, 16
colInfo *(*colInfo-class*)*, 5
colInfo-class, 5
colName *(*colInfo-class*)*, 5
colName,colInfo-method *(*colInfo-class*)*, 5
colName<- *(*colInfo-class*)*, 5
colName<-,colInfo-method *(*colInfo-class*)*, 5
colType *(*colInfo-class*)*, 5
colType,colInfo-method *(*colInfo-class*)*, 5
colType<- *(*colInfo-class*)*, 5
colType<-,colInfo-method *(*colInfo-class*)*, 5
convert2PData *(*importPhenoData*)*, 18
createPData *(*importPhenoData*)*, 18

dataViewer, 6, *31*
dbArgsWidget, 7
DPExplorer, 8, *9*
dropArgs, *33*
dropArgs *(*setArgsList*)*, 32

dropColumn, *34*
dropColumn *(*setArgsList*)*, 32
dropOrNot *(*colInfo-class*)*, 5
dropOrNot,colInfo-method
*(*colInfo-class*)*, 5
dropOrNot<- *(*colInfo-class*)*, 5
dropOrNot<-,colInfo-method
*(*colInfo-class*)*, 5

eExplorer, *9*, 9, *10*, *28*

fileBrowser, *3*, 10, *13*, *21, 22*, *29*, *42*
fileWizard, *4*, 12, *17*
find.type, *16*
find.type *(*guess.sep*)*, 16
finish, *21, 22*
finish *(*importWizard*)*, 20
formals, *4*
formatArg, *4, 5*
formatArg *(*argsWidget*)*, 4
funcs2Char, *5*
funcs2Char *(*argsWidget*)*, 4

getAFrame, *21, 22*
getAFrame *(*importWizard*)*, 20
getArgs, *33, 34*
getArgs *(*setArgsList*)*, 32
getColInfo, *33, 34*
getColInfo *(*setArgsList*)*, 32
getCovarDesc *(*importPhenoData*)*, 18
getCState, *33, 34*
getCState *(*setArgsList*)*, 32
getExclude *(*pExplorer*)*, 27
getExCode, *10*
getExCode *(*eExplorer*)*, 9
getFileContents, *28*
getFileContents *(*pExplorer*)*, 27
getHelpFile, *10*
getHelpFile *(*eExplorer*)*, 9
getLightTW, 13
getLineData *(*setArgsList*)*, 32
getMoreArgs, *33, 34*
getMoreArgs *(*setArgsList*)*, 32
getName4Data, *22*
getName4Data *(*importWizard*)*, 20
getOBJWidget *(*importPhenoData*)*, 18
getPkgContents, *28*
getPkgContents *(*pExplorer*)*, 27
getPWidget, *5*
getPWidget *(*argsWidget*)*, 4
getRowNames *(*guess.sep*)*, 16
getRPkgs, *28*
getRPkgs *(*pExplorer*)*, 27
getShowNum, *33, 34*
getShowNum *(*setArgsList*)*, 32
getSNCNums *(*importPhenoData*)*, 18
getState1Frame, *21, 22*
getState1Frame *(*importWizard*)*, 20
getState2Frame, *22*
getState2Frame *(*importWizard*)*, 20
getState3Frame, *22*
getState3Frame *(*importWizard*)*, 20
getSymbol, *5*
getSymbol *(*argsWidget*)*, 4
getTopCan, *21, 22*
getTopCan *(*importWizard*)*, 20
getTopLevel, *9*
getTopLevel *(*DPExplorer*)*, 8
getTrueNullNa, *5*
getTrueNullNa *(*argsWidget*)*, 4
getWvalues, 14
guess.header *(*guess.sep*)*, 16
guess.sep, *16*, 16, *33*

hasChar, 17
hasDesc, *28*
hasDesc *(*pExplorer*)*, 27
hasPrefix, *11, 12*, *29*
hasPrefix *(*hasChar*)*, 17
hasSuffix, *11, 12*, *29*
hasSuffix *(*hasChar*)*, 17

importPhenoData, 18
importWizard, *5–7*, 20, *21*, *34*
initImportWizard, *21*
initImportWizard *(*importWizard*)*, 20

list, *25*, *42, 43*
list.files, *28*
listSelect, 23, *27*, *34, 35*
loadDataPkg, *9*
loadDataPkg *(*DPExplorer*)*, 8

makePhenoData *(*importPhenoData*)*, 18
MIAME, *36*
moreArgs, *34*
moreArgs *(*setArgsList*)*, 32

noAuto *(*pickObjs*)*, 31

objectBrowser, 24, *25, 26*, *31*, *42*
objExists *(*importPhenoData*)*, 18
objNameToList, 25
objViewer, *23, 24*, 26

pExplorer, 27, *28*
pickFiles, *3*, *12*, *18*, 29
pickItems, 30
pickObjs, 31
popStartLine *(*importWizard*)*, 20
print.pWidget *(*values.Widget*)*, 38
print.Widget *(*values.Widget*)*, 38
procHelp, *28*
procHelp *(*pExplorer*)*, 27
procHTML, *28*
procHTML *(*pExplorer*)*, 27
procPDF, *28*
procPDF *(*pExplorer*)*, 27
procRda, *28*
procRda *(*pExplorer*)*, 27

read.table, *12*, *16*, *21, 22*, *34*
readFileByLines *(*importWizard*)*, 20
readLines, *21*, *33*

setArgsList, 32, *33*
setColInfos, *33*
setColInfos *(*setArgsList*)*, 32
setColName, *34*
setColName *(*setArgsList*)*, 32
setColType, *34*
setColType *(*setArgsList*)*, 32
setNewState, *33*
setNewState *(*setArgsList*)*, 32
setQuote, *22*
setQuote *(*importWizard*)*, 20
setQuoteList, *22*
setQuoteList *(*importWizard*)*, 20
setSepRadios, *22*
setSepRadios *(*importWizard*)*, 20
setSkip, *33*
setSkip *(*setArgsList*)*, 32
setState1BFrame, *21, 22*
setState1BFrame *(*importWizard*)*, 20
setState1MFrame, *22*
setState1MFrame *(*importWizard*)*, 20
setState1TFrame, *21*
setState1TFrame *(*importWizard*)*, 20
setState2BFrame, *22*
setState2BFrame *(*importWizard*)*, 20
setState2MFrame, *22*
setState2MFrame *(*importWizard*)*, 20
setState3BFrame, *22*
setState3BFrame *(*importWizard*)*, 20
setState3TFrame, *22*
setState3TFrame *(*importWizard*)*, 20
showData4State1, *22*
showData4State1 *(*importWizard*)*, 20
showData4State2, *22*
showData4State2 *(*importWizard*)*, 20
stdType, 34
stdView *(*stdType*)*, 34

tkMIAME, 35
tkphenoData, 37
tkSampleNames, 37
TRUE, *18*

values.Widget, 38
vExplorer, *10*, 40
viewVignette *(*vExplorer*)*, 40

WbuttonFun *(*WName*)*, 43
WbuttonText *(*WName*)*, 43
WcanEdit *(*WName*)*, 43
WEnd *(*WName*)*, 43
WfromText *(*WName*)*, 43
whatDeli, *33, 34*
whatDeli *(*setArgsList*)*, 32
widgetRender, *15*, *38, 39*, 41, *44*
WLValue *(*WName*)*, 43
WLValue<- *(*WName*)*, 43
WName, 43
WpostFun *(*WName*)*, 43
WpreFun *(*WName*)*, 43
WRButtons *(*WName*)*, 43
writeBut *(*listSelect*)*, 23
writeCol4Matrix, *22*
writeCol4Matrix *(*importWizard*)*, 20
writeLabel *(*listSelect*)*, 23
writePDRowNames *(*importPhenoData*)*, 18
writePhenoTable *(*importPhenoData*)*, 18
writeSelBox *(*listSelect*)*, 23
WtoText *(*WName*)*, 43
WValue *(*WName*)*, 43
WValue<- *(*WName*)*, 43
WwList *(*WName*)*, 43
WwList<- *(*WName*)*, 43