Package ‘RGtk2Extras’

February 19, 2015

Type    Package
Title   Data frame editor and dialog making wrapper for RGtk2
Version 0.6.1
Date    2012-09-26
Author  Tom Taverner, with contributions from John Verzani, Iago Conde
        and Liviu Andronic
Maintainer Tom Taverner <t.taverner@gmail.com>
Description Useful add-ons for RGtk2
Depends methods, RGtk2(>= 2.12.8)
Recommends cairoDevice
Suggests gWidgetsRGtk2
License GPL (>= 2)
LazyLoad yes
Repository CRAN
Date/Publication 2012-10-23 09:46:22
NeedsCompilation yes

R topics documented:

dfedit ......................................................... 2
dfview ....................................................... 3
get.value ....................................................... 4
GetTaskPasteIn ............................................... 4
get_all_tables ............................................... 5
gtkDfEdit ..................................................... 5
gtkDfEditCommandData ...................................... 10
gtkDfEditDoTask ............................................. 11
gtkDfEditEditFactors ....................................... 13
gtkDfEditGetColumnNames .................................. 13
gtkDfEditGetDataFrame ..................................... 14
gtkDfEditGetDatasetName .................................. 14
**Description**

Convenience function for editing a data frame in a window.

**Usage**

```r
dfedit(items,
    dataset.name = deparse(substitute(items)),
    size = c(600, 300), col.width=64,
    editable=TRUE,
    autosize = is.null(dim(items))||ncol(items)<25,
    update=TRUE, modal=TRUE)
```

**Arguments**

- `items` The data frame to edit
- `dataset.name` Optional name to assign the dataset when updating
- `size` Optional window size request
- `col.width` Optional column size
- `editable` Allow user editing of the frame
- `autosize` Columns automatically size to fit headers
- `update` Update the frame on the fly with editing
- `modal` TRUE for the editor to halt R and return an object
**dfview**

**Details**

IF set to NULL, is the name of the dataset.

**Value**

Returns the edited data frame after running modally

---

**dfview**  
*Convenience function for viewing a data frame in its own window*

**Description**

Convenience function for viewing a data frame in a window.

**Usage**

```r
dfview(items,
       dataset.name = deparse(substitute(items)),
       size=c(600, 300), col.width=64,
       editable=FALSE,
       autosize = is.null(dim(items)) || ncol(items) < 25,
       update=FALSE)
```

**Arguments**

- `items`  
  The data frame to edit
- `dataset.name`  
  Optional name to assign the dataset when updating
- `size`  
  Optional window size request
- `col.width`  
  Optional column size
- `editable`  
  Allow user editing of the frame
- `autosize`  
  Columns automatically size to fit headers
- `update`  
  Update the data frame on the fly

**Details**

IF set to NULL, is the name of the dataset.

**Value**

Returns the edited data frame after running modally
get.value  \hspace{2cm} \textit{Return all tables}

\textbf{Description}

generic value setting method

\textbf{Usage}

get.value(dlg.item, ...)

\textbf{Arguments}

\begin{itemize}
  \item \texttt{dlg.item} \hspace{0.5cm} \texttt{gtkWidget}
  \item \texttt{...} \hspace{0.5cm} \texttt{args}
\end{itemize}

\textbf{Details}

This function can be called on dialogItems to get the value they are set to.

---

\textbf{GetTaskPasteIn} \hspace{2cm} \textit{Return all tables}

\textbf{Description}

This function returns a list of all tables in .GlobalEnv, including those within lists.

\textbf{Usage}

\texttt{GetTaskPasteIn(theframe, dat, insert.row, insert.col,}
\texttt{ do.rownames=F, do.colnames=F,}
\texttt{ do.coercion=T)}

\textbf{Arguments}

\begin{itemize}
  \item theFrame \hspace{0.5cm} Dataset to use
  \item dat \hspace{0.5cm} insertion frame
  \item insert.row \hspace{0.5cm} row to insert
  \item insert.col \hspace{0.5cm} col to insert
  \item do.rownames \hspace{0.5cm} do rownames
  \item do.colnames \hspace{0.5cm} do colnames
  \item do.coercion \hspace{0.5cm} coercion
\end{itemize}
get_all_tables

Details
This is a helper function used to generate a task list to allow reversible pasting of data into the frame.

get_all_tables  Return all tables

Description
This function returns a list of all tables in .GlobalEnv, including those within lists.

Usage
get_all_tables()

gtkDfEdit  gtkDfEdit

Description
An RGtk2 spreadsheet package for editing data frames. Improves on base edit.data.frame function found in utils

Usage
gtkDfEdit(items, dataset.name = deparse(substitute(items)),
size.request=c(600, 300), col.width=64,
dataset.class = "data.frame", editable = TRUE,
autosize = length(dim(items)) < 2 || ncol(items)<25,
update=TRUE, envir = .GlobalEnv, ...)

Arguments
items  The data frame to edit
dataset.name  The name of the data frame object to modify.
size.request  The size request for the window.
col.width  The column width.
dataset.class  Class to coerce data object in global environment to (frame or matrix)
editable  Allow user editing of data in frame.
autosize  Columns automatically size to fit headers
update  Update data frame on the fly with editing.
envir  Environment to assign data frame into on editing
...  Additional args, ignored
Details

gtkDfEdit is an RGtk2 based data frame viewer and editor widget intended to be familiar to spreadsheet users and to form part of larger GUI projects. It provides a way to edit a data frame or matrix (see Notes for a description).

Changes made in the spreadsheet will appear instantly in the data frame.

The "[" method is used for data-frame like extraction from the object.
The "[<-" method is used for assignment to the table.
The $getSelection method returns a list of selected row and column indices.
The $getModel method returns the backing RGtkDataFrame.
The $getDimension method returns the backing data frame dimension.
The $getColumnNames method returns the column names.
The $getRowNames method returns the row names.
The $setColumnName(idx, new.name) method sets the column name at a particular index.

Value

A GtkContainer containing the widget.

Note

The editor consists of row names, column names, the main grid of cells, and the left-hand corner cell. You can move around within the grid using the keyboard, the scrollbars, or by clicking and dragging with the mouse.

Navigation Around The Grid

Keyboard navigation uses the familiar arrow or Shift, Shift-Enter, Tab, Shift-Tab, PgUp, PgDown, Ctrl-PgUp, Ctrl-PgDown, Home, End keys. These work when either the grid or the column of row names has the focus.

Pressing a non-navigation key when the row names have focus will cause automatic navigation to the closest match for the row name. The name matching entry dialog will go away after a couple of seconds.

Mouse navigation to a grid location can be done via the scroll bars on the grid or using the scroll wheel.

Editing The Grid

Using non-navigation keys in a selected cell will start editing within the cell. If the column is of factor type, the cell entry will provide the user with an autocompletion containing existing factor levels.

Focusing out of the cell or pressing any navigation key will end the edit. Edited cell entries will be coerced to the column’s data type, so alphabetical strings put into numeric columns will turn into a platform-dependent variant of "NA". All character strings are stripped of beginning and end whitespace. Adding a new item to a factor column will automatically update factor levels.

Deleting cells sets their contents to either NA, or "" if the cells are of type character. Deleting cell contents can be done in a number of ways. Deleting a block of selected cells on the grid can be
done by selecting them, then pressing the Delete key. The Backspace key deletes the cell the cursor
is on ignoring all selections.

Deleting entire rows and columns can be done by selecting the row names or column headers then
pressing Delete, or else by right-clicking on the row names or column headers to bring up the
context menu then clicking "Clear Contents".

Changes made in the data frame editor are automatically and invisibly updated in the linked R data
frame object. However, changes made in the linked data frame object are NOT updated in the grid
display.

Ctrl-Z undoes any editing action on the grid. Actions that have side effects on the data, such as
correction, are not fully undoable, which reflects the way R handles these functions.

Editing Row And Column Names

Double clicking row names and column names allows the user to edit them. Typing in the replace-
ment name and pressing Enter, Escape or clicking somewhere else will set the changed row or
column name.

Duplicate row names will be turned into unique values by replacing each duplicate with the lowest
possible ordinal number.

Editing The Data Frame Object Name

The name of the data frame object is displayed in the top-left corner cell.

Double clicking the top-left corner cell allows the data set to be updated and reassigned when Return
is pressed. When editing is finished the data frame in the editor will be written to the new dataset
name.

Cell Selection

Active cells or cell selections are indicated with a focus rectangle. Active columns are indicated by
a colored highlight. By clicking and dragging with the mouse, you can scroll around the grid in two
dimensions and select a rectangular block of cells. Alternatively, you can use the keyboard arrow
keys with Shift held down to select a block.

Left-clicking and dragging on a region of cells selects the region and draws a focus rectangle around
it. Selections are indicated by highlighted rows, column headers and a drawn focus rectangle. Rows
can be selected by focusing on the row name column then doing either mouse or keyboard selection.

The keyboard can also be used for grid selection. Left clicking on column headers or row names
selects the columns or rows. Multiple, or ranges of, columns or rows can be selected using the usual
Ctrl-Click and Shift-Click combinations.

Macintosh users should use Ctrl-Command-Click instead of Ctrl-Click.

Ctrl-A, or clicking the top-left corner cell, selects all cells on the grid.

Copying And Pasting

Copied and pasted data is in tab-delimited form and can be pasted directly into other spreadsheets
or text editors. We use the usual platform specific line separator.

In Linux, the functions xclip and xsel must be available at the command line for copy and paste to
work. In Mac, pbcopy and pbpaste are used. In Windows, we use the R functions writeClipboard
and readLines.

Ctrl-V pastes cell selections to the clipboard at the selected point into a block defined by the size
of the pasted matrix and starting at the top left corner of the top left selected cell. If necessary, this
operation will change the dimension of the grid. Pasting automatically coerces data to the type in the column.

Ctrl-C entered while focus is on the grid copies the selected block of cells.

Alternatively these functions can be accessed from the grid right click context menu "Copy" and "Paste".

Copying a cell block into the clipboard will not include row or column names. To include row and column names in the copy operation, select "Copy With Names" from the grid right click context menu.

Copying and pasting rows and columns can be done through the right click context menus over row headers or column headers in the "Copy" and "Paste" commands. Copying from a column will include the column header and copying from a row will include the row header. Pasting on columns will update the column headers.

Copying an entire data frame into the editor can be done through the grid top left corner right click selection menu through the "Paste..." command. This command brings up a global paste dialog which allows the user to choose whether the pasted data has row names and/or column names. When "OK" is pressed the data will be pasted in.

**Data Coercion And Special Functions**

From the right click context menu on column headers the selected data frame columns’ assigned type can be changed. Available data types are Numeric, Integer, Logical, Character, Factor. Factor is a special enumerated data type (also known as a category) which can have its attributes set using the in-built Factor Editor (see below). To coerce a data column, just open this menu and click the desired type.

Factors can be coerced differently; either to the values of their levels or to the integer ordinal value of those levels.

The column context menu function "Set As Row Names" sets the contents of the column as the data frame’s row names. The menu function "Shorten Names..." replaces long string names with their unique abbreviations.

Right clicking the top-left corner cell selects all cells and brings up a menu allowing global cut, copy, and paste actions. "Edit Dataset Name" allows the data set name in the R environment to be reassigned. "Default Row Names" sets the row names to their ordinal numbers from 1 to the number of rows. "Default Column Names" sets the column names to the familiar spreadsheet-style defaults.

Coercion can be partially undone via Ctrl-Z, but to reflect R’s handling of coercion, coercing between classes that are not interchangable, such as from a character to a numeric variable, is not undoable.

**Inserting And Deleting Columns And Rows**

Right clicking on row name headers brings up a menu which allows Insert and Delete actions on data columns. "Insert" inserts a blank row before the row clicked. "Delete" deletes the selected row range and is not available when rows are not selected.

Right clicking on column name headers similarly brings up a menu which allows Insert and Delete actions on data columns. "Insert" inserts a blank column before the column clicked. To insert a blank column at the end, click the blank header at the right hand side. "Delete" deletes the selected column(s).

**Editing Factors**
Right clicking on a column header of a factor column, then selecting "Factor Editor", or right clicking a selected factor column, opens the Factor Editor which allows factor levels, order and contrasts to be set.

The Factor Editor window displays the choice of data frame factor columns, the factor levels of the selected columns, and the contrasts in the "Factor Contrasts" expander. When a column is selected, if it is a factor, its levels are displayed in the "Factor Level Order" frame. The factor levels can be re-ordered, edited, deleted or additional levels added by using the buttons to the right of the level display.

Factors are associated with contrast matrices for use in analysis of variance and regression models. The Factor Editor allows contrasts to be set by opening the "Factor Contrasts" expander frame and selecting the desired contrast type. The default contrast type sets the first ordered level as the control.

It is often desirable to fill in factor levels according to a pattern, for example, in specifying a balanced experimental design. This can be done in two ways. First, highlighting a region of cells then right clicking on a Factor column, pulls up the context menu including three options, "Fill Selected Down" "Randomize Selected", "Fill In Blocks".

"Fill Selected Down" fills all selected cells in the column with the FIRST selected cell.

"Randomize Selected" replaces all selected cells within the column that was clicked with the same contents, in randomized order.

"Fill In Blocks" opens a new window containing a spin button specifying the block size of factor level repeats to fill the selected region. For example, factor levels A, B, C, block size 2, the region is filled down A, A, B, B, C, C, A, A, B, B, C, C, etc. The region will be filled when the spin button is modified or Enter is pressed, and the fill can be cancelled by pressing Cancel. The OK button will cause the changes to be fixed.

The same factor filling options as described above can be accessed directly from the Factor Editor window, which can be called up as described above using "Selected", "Random Fill" and "Fill with Replicates...". In this case, it fills the entire column, not just the highlighted region.

Sorting Data
From the right-click menu on the corner left hand cell or on the columns, the "Sort..." dialog can be opened. This dialog consists of (1) a "Sort Key" Selection frame (2) "Add/Remove Key" frame to add/remove sort keys (3) "OK" and "Cancel" buttons.

Sort operations on the data can be undone via Ctrl-Z.

The "Sort Key" frame contains key choice items consisting of a combo box for key selection, radio buttons for coercion of the key, and radio buttons for choosing the sort direction. Sorting starts with the first key, breaking ties by keys further down the list.

The combo box allows the user to choose the column of the data frame, including the row names, they wish to sort on.

The coercion radio buttons allow the user to sort on the corresponding column by the default xtfrm ranking, or by first coercing to character or numerical form. This can be useful for sorting numeric row names or factors.

The "Ascending" and "Descending" radio buttons choose whether the sort on the corresponding key item is in ascending or descending order.

The "Add/Remove Keys" frame contains a button "Add A Key" allowing the user to add another key choice item to the "Sort Key" frame and a button "Remove A Key" to remove the last key choice item in the frame. There is no limit to the number of keys that can be sorted.
Finally the "OK" button initiates the data frame sort and the Cancel button closes the dialog.

Author(s)

Tom Taverner <t.taverner@gmail.com>, with contributions from John Verzani

See Also

dfedit

Examples

```r
win <- gtkWindowNew()
obj <- gtkDfEdit(iris)
win$add(obj)

obj[1,1,drop=FALSE]
obj$getSelection()
```

---

**gtkDfEditCommandData**

*Open the Command Editor*

**Description**

Applies a string command

**Usage**

```r
gtkDfEditCommandData(x)
```

**Arguments**

- `x` string

**Details**

This function opens the command editor which allows R commands to be applied to a selected data submatrix. See ?gtkDfEdit
Description

Implement defined spreadsheet actions

Usage

gtkDfEditDoTask(x, task)

Arguments

x
The RGtk2DfEdit object

task
The task list to implement.

Note

An action item is a list containing the action function name and its arguments:
list(func=action.name, arg=list(arg1=value1, arg2=value2, ...))
A task is a list of one or more action items.

When the task is passed to x$doTask the backing data frame will be updated sequentially with each
action and the model updated after the action list is complete. This way, sequences of commands
can be built up, performed and undone in a single step.

The following action names and function argument lists are available.
ChangeCells: function(nf, row.idx, col.idx, do.coercion=T)
SetFactorAttributes: function(idx, info)
CoerceColumns: function(theClasses, idx)
ChangeColumnNames: function(theNames, idx)
ChangeRowNames: function(theNames, idx)
DeleteRows: function(idx)
InsertRows: function(nf, idx)
InsertNARows: function(idx)
DeleteColumns: function(idx)
InsertColumns: function(nf, idx)
InsertNAColumns: function(idx, NA.opt="")
nf is the new data frame being passed to the function, if any.
do.coercion is the flag which tells the editor whether to coerce the new frame (nf) to the type of the
old data frame or not.
theClasses and theNames are the new classes or new names being applied to the function.
idx is the indices at which to insert or change new columns or rows, or column or row names.
theNames and theClasses must have the same length as idx, and when "nf" is present nf must have the same number of rows as idx if InsertColumns is called, and the same number of columns as idx if InsertRows is being called.

info is a list of form list(levels, contrasts, contrast.names). contrasts and contrast.names may or may not be present.

NA.opt is an optional NA to pass to InsertNAColumns to coerce to a particular type, for example NA.opt=NA_real_ will make the NA columns inserted numeric.

Examples

```
win = gtkWindowNew("gtkDfEdit Demo")
obj <- gtkDfEdit(iris)
win$add(obj)
win$show()

task <- list(
    list(func="ChangeCells",
      arg=list(nf=array(4, c(2,2)), row.idx=1:2, col.idx=1:2))
)
obj$doTask(task)

task <- list(
    list(func="InsertRows",
      arg=list(nf=iris[,1], row.idx=1))
)
obj$doTask(task)
obj$undo()

task <- list(
    list(func="InsertColumns",
      arg=list(nf=iris[,1], col.idx=1))
)
obj$doTask(task)
obj$undo()

task <- list(
    list(func="InsertNARows", arg=list(row.idx=2)),
    list(func="InsertNAColumns", arg=list(col.idx=2))
)
obj$doTask(task)
obj$undo()

task <- list(
    list(func="ChangeRowNames",
      arg=list(theNames=c("hi", "there"), row.idx=1:2))
)
```
gtkDfEditEditFactors  

Description
Open the Factor Editor, see gtkDfEdit

Usage
gtkDfEditEditFactors(x)

Arguments

x 

The RGtk2DfEdit object

Value

Nothing

gtkDfEditGetColumnNames

Return the columns of the RGtk2DfEdit object...

Description

Return the columns of the RGtk2DfEdit object

Usage

gtkDfEditGetColumnNames(object)
Arguments

object The RGtk2DfEdit object

Value

Returns the column names for the current object

gtkDfEditGetDataFrame

Return a data frame from the RGtk2DfEdit object...

Description

Return a data frame from the RGtk2DfEdit object

Usage

gtkDfEditGetDataFrame(object)

Arguments

object The RGtk2DfEdit object

Value

Returns the data frame with row names and column names

gtkDfEditGetDatasetName

Return all tables

Description

Gets dataset name

Usage

gtkDfEditGetDatasetName(x)

Arguments

x The S3 object

Details

This method returns the name of the gtkDfEdit dataset.
gtkDfEditGetDimension

*Return the dimensions (nrow, ncol) of the RGtk2DfEdit object...*

**Description**

Return the dimensions (nrow, ncol) of the RGtk2DfEdit object

**Usage**

`gtkDfEditGetDimension(object)`

**Arguments**

object The RGtk2DfEdit object

**Value**

Returns the number of rows and columns – not counting row names

-----------------------

gtkDfEditGetModel

*get Model from object...*

**Description**

get Model from object

**Usage**

`gtkDfEditGetModel(object)`

**Arguments**

object The RGtk2DfEdit object

**Value**

the RGtk2DataFrame that is the backend model for the widget
gtkDfEditGetRowNames  Return the row names of the RGtk2DfEdit object...

Description
Return the row names of the RGtk2DfEdit object

Usage
gtkDfEditGetRowNames(object)

Arguments
object  The RGtk2DfEdit object

Value
Returns the row names for the current object

gtkDfEditGetSelection  get selected row and column indices...

Description
get selected row and column indices

Usage
gtkDfEditGetSelection(object)

Arguments
object  The RGtk2DfEdit object

Value
the 1-indexed selected rows
gtkDfEditSetActionHandler

Setting user defined functions on the editor

Description

Set a user function to call when some action is performed

Usage

`gtkDfEditSetActionHandler(object, func.name, handler=NULL, data=NULL)`

Arguments

- **object**: The RGtk2DfEdit object
- **func.name**: The name of the spreadsheet action which triggers the function.
- **handler**: Function to call when the action occurs. Signature varies, see below. If NULL (default) no handler is called.
- **data**: Optional data to pass to the function.

Details

IF set to NULL, no handler is called.

Note

The following action names and function signatures for the handler can be used. "Selection" means a cell range is selected and a selection rectangle is drawn.

Selection: function(rows, cols, data=NULL)

ChangeCells: function(obj, nf, row.idx, col.idx, do.coercion=T, data=NULL)

SetFactorAttributes: function(obj, idx, info, data=NULL)

CoerceColumns: function(obj, theClasses, col.idx, data=NULL)

ChangeColumnNames: function(obj, theNames, col.idx, data=NULL)

ChangeRowNames: function(obj, theNames, row.idx, data=NULL)

DeleteRows: function(obj, row.idx, data=NULL)

InsertRows: function(obj, nf, row.idx, data=NULL)

InsertNARows: function(obj, row.idx, data=NULL)

DeleteColumns: function(obj, col.idx, data=NULL)

InsertColumns: function(obj, nf, col.idx, data=NULL)
InsertNAColumns: function(obj, col.idx, NA.opt="", data=NULL)
obj is the gtkDfEdit object being edited.
data is optional user data to pass to the function.
"nf" is the new data frame being passed to the function, if any.
doa.coercion" is the flag which tells the editor whether to coerce the new frame (nf) to the type of the
old data frame or not.
theClasses and theNames are the new classes or new names being applied to the function.
row.idx and col.idx are the row and column indices where the action occurred.
idx is the row or column index where the action occurred, for some actions which have only one
kind of index.
theNames and theClasses must have the same length as idx, and when "nf" is present nf must have
the same number of rows as idx if InsertColumns is called, and the same number of columns as idx
if InsertRows is being called.
info is a list containing factor information of form list(levels, contrasts, contrast.names, is.ordered).
contrasts and contrast.names may or may not be present.
NA.opt is an optional NA to pass to InsertNAColumns to coerce to a particular type, for example
NA.opt=NA_real_ will make the NA columns inserted numeric.

Examples

win = gtkWindowNew("gtkDfEdit Demo")
obj <- gtkDfEdit(iris)
win$add(obj)
win$show()

obj$setActionHandler("ChangeCells",
  handler=function(obj, nf, row.idx, col.idx, do.coercion)
    print(paste("Cells changed at R",
      if(!missing(row.idx)) row.idx, ", "C",
      if(!missing(col.idx)) col.idx, sep=""))
)

obj$setActionHandler("SetFactorAttributes",
  handler=function(obj, col.idx, info) {
    print(paste("factor changed at", col.idx,
      "new levels", paste(info$levels,
      collapse=", "))
  })
)

obj$setActionHandler("CoerceColumns", function(obj, theClasses, col.idx)
  print(paste("columns", col.idx,
    "of", obj$getName(), "coerced to",
    theClasses))
)

obj$setActionHandler("ChangeColumnNames", function(obj, theNames, col.idx) {
  print(paste("column names at", col.idx,
    "changed to", theNames))
)
gtkDfEditSetDatasetName

))

obj$setActionHandler("ChangeRowNames", function(obj, theNames, row.idx) {
    print(paste("row names at", row.idx, "changed to", theNames))
})

obj$setActionHandler("DeleteRows", function(obj, row.idx) {
    print(paste("rows at", row.idx, "deleted"))
})

obj$setActionHandler("InsertRows", function(obj, nf, row.idx) {
    print(paste("rows inserted at", row.idx))
    print(nf)
})

obj$setActionHandler("InsertNARows", function(df, row.idx) {
    print(paste("rows inserted at", row.idx))
})

obj$setActionHandler("DeleteColumns", function(obj, col.idx) {
    print(paste("columns at", col.idx, "deleted"))
})

obj$setActionHandler("InsertColumns", function(obj, nf, col.idx) {
    print(paste("cols inserted at", col.idx))
})

obj$setActionHandler("InsertNAColumns", function(obj, nf, col.idx, NA.opt) {
    print(paste("cols inserted at", col.idx))
})

obj$setActionHandler("Selection", function(obj, row.idx, col.idx) {
    print(paste(paste(paste(length(row.idx), "R", sep=""), "x", paste(length(col.idx), "C", sep=""))))
})

obj$setActionHandler("RowClicked", function(obj, idx) print(obj[idx,]))

obj$setActionHandler("ColumnClicked", function(idx, data)
    print(obj[,idx]))

---

gtkDfEditSetDatasetName

Return all tables

Description

Sets name
Usage

gtkDfEditSetSetName(x, new.name)

Arguments

x x
new.name new.name

Details

Sets the name of the data set to new.name, in .GlobalEnv

Description

Opens the Sort dialog, see gtkDfEdit

Usage

gtkDfEditSort(x)

Arguments

x The RGtk2DfEdit object

Value

Nothing

Description

Undoes the previous action done in the spreadsheet

Usage

gtkDfEditUndo(x)

Arguments

x The RGtk2DfEdit object
**my_choose_files**

**Value**

Nothing

**Description**

Platform independent file chooser

**Usage**

```r
my_choose_files(fn = ".*", multi=FALSE, filters=NULL, type = "open", caption=NULL)
```

**Arguments**

- `fn`: filename
- `multi`: multiple
- `filters`: Win filters
- `type`: type of dialog
- `caption`: optional dialog caption

**Details**

This is a wrapper which allows the user to select a file under Windows, Mac and Linux. The type can be "open", "save" or "selectdir". Multiple selectdir isn’t supported.

---

**object.exists**

**Description**

Does object of form a\$b exist

**Usage**

```r
object.exists(text, envir=parent.env(environment()))
```
Arguments

- text: text to evaluate
- envir: environment

Details

Does object of form a\$b exist

---

quick_message Return all tables

---

Description

Create a modal message box.

Usage

quick_message(message="", caption="Warning", win=NULL)

Arguments

- message: message to use
- caption: caption for window
- win: gtkWindow

Details

Create a quick message box.

---

run.dialog Simple Dialog Maker package for RGtk2

---

Description

An RGtk2/gWidgetsRGtk2 based package for quick and easy dialog development.
run.dialog

Usage

run.dialog(func, pos = -1, envir = as.environment(pos), dlg.list = tryCatch(
  get(paste(func.name, "dialog", sep=".")),
  error = function(e) NULL),
  var.browser=NULL, parent.window=NULL, auto.assign=TRUE,
  do.long.running=FALSE, OK_handler= default.handler,
  output.name.rule = "append", output.name = NULL,
  do.logging = TRUE, log.handler = NULL,
  user.args = NULL, ...
)

Arguments

func Function to be run when "OK" is pressed
pos where to look for the dialog markup (see the details section)
envir alternative specification of environment
dlg.list Optional list containing dialog markup
var.browser An optional gWidgets gvarbrowser
parent.window An optional parent window to set modal
auto.assign Assign the function return value, if it exists, to the global environment with an automatically assigned name
output.name.rule Rule for assigning output name
do.long.running Use long running task routine for Windows
OK_handler Function to pass do.call(func, args to)
output.name Name of return value; overrides auto.assign and output.name.rule
do.logging Send the deparsed function call to an external function?
log.handler Function to take the deparsed function call
user.args Optional values to pass to the dialog which overrule those in dialog
... Additional arguments
Details

This package is intended to assist rapid development of simple dialogs to serve as front-ends to existing R functions. It is a riff off (or rip off) of ideas from John Verzani’s traitr toolkit where flexibility is sacrificed for ease of use. Complex layouts are not supported; this package is for getting things working quickly, not necessarily looking beautiful.

To create a working dialog that calls a function, a corresponding markup list is needed specifying how each argument of the function should look on the screen - in other words, what widgets to use. Then run.dialog() can be called on the function.

Calling run.dialog()

If your function is called MyFunction and the markup list is called MyFunction.dialog or .MyFunction.dialog then calling run.dialog(MyFunction) will attempt to locate the corresponding markup list and create and show the dialog. When the user presses "OK" the dialog will call the function with the specified arguments.

The dialog will be modal and centered on the gtkWindow parent.window if this is specified. The R main loop will be blocked until the dialog is destroyed.

Alternatively, run.dialog(MyFunction, dlg.list=MyMarkupList) will explicitly specify the dialog markup list for MyFunction.

Return Values

run.dialog() returns a list containing retval and args, where retval is the return value of the function call and args is a list containing the named arguments specified from the dialog.

Dialog Markup Specification

For a function MyFunction with signature

MyFunction = function(arg1, arg2, ...)

the dialog markup list is specified by

MyFunction.dialog = list(
   dialog_markup = value0, ..., 
   arg1.dialogItem1 = value_of_arg1, markup1 = value1, ..., 
   arg2.dialogItem2 = value_of_arg2, markup2 = value2, ..., 
   ...
)

The only necessary items in the markup list are those of form

argn.dialogItem = value_of_argn

The name arg1.dialogItem in the MyFunction.dialog markup list tells run.dialog() to display a widget of type dialogItem corresponding to arg1. This widget is initialized to value_of_arg1 and passes this value as the arg1 parameter to MyFunction when "OK" is clicked.

Optional markup elements that affect the entire dialog are specified by dialog_markup items before the first argNdialogItem, and markup elements affecting individual dialogItems are specified between them and the next dialogItem (or in the case of the last item, the end of the list).

dialogItem can be any one of the following: integerItem, numericItem, rangeItem, stringItem, true-FalseItem, choiceItem, radiobuttonItem, objectItem, dataframeItem, fileItem, variableSelectorItem,
listItem, buttonItem. Each of these Items displays a different commonly used dialog GUI widget which are listed below.

Most widgets are also set into a small rectangular frame which contains a label. By default the label is set to the name of the corresponding function argument, but this can be set using the label markup.

The widgets are laid out very simply within the dialog GUI, set top to bottom in a column. If the column layout is told to break using the BREAK=TRUE markup, another column is added to the left hand side and the widgets after that point are laid out into the new column. An optional label for the entire dialog can be added to the top over all columns.

**Markup**

**Markups affecting the entire dialog**

These are set before the first arg.dialogItem

- **label** = "This GUI does x" sets the label centered over all columns.

- **title** = "My Function" sets the title of the dialog window.

- **keep.open** = TRUE tells the dialog to stay open after "OK" is pressed. This can be useful if your function plots something, and the user is likely to want to try different plot settings.

- **show.progress** = TRUE tells run.dialog that the function should display a dialog while it is executing to allow monitoring of progress and cancellation. This is based on Felix Andrews’ interrupt.c from playwith. If this markup is present then when "OK" is pressed a small modal Progress dialog will appear with a cancel button. When this button is pressed, an interrupt signal is sent and the Progress dialog is destroyed and the main dialog is destroyed if appropriate. This is not guaranteed to stop all running R processes. Moreover, if your function arguments contain progressbar or progresslabel, then run.dialog will pass your function a gtkProgressBar and a gtkLabel respectively, to allow user monitoring of progress via progressbar$setFraction() or progressbar$setText(). See ?gtkProgressBar and ?gtkLabel.

  Note 1: if your function updates the progress bar or progress label, it should give these arguments default NULL values and check they are not missing before using them, otherwise it will break if run without a dialog.

  Note 2: if your dialog markup (as opposed to your function) contains progressbar or progresslabel arguments, then run.dialog will not pass those widgets to your function automatically.

- **long.running** = TRUE tells run.dialog that the function will run for a long time before returning. This is fairly experimental, and doesn’t work anywhere but Windows yet. In Windows, it will do the following things:

  1. Save the R session to the current working directory.
  2. Create a batch file containing a command to load the R session, delete the session file, carry out the function call, and a command to save the session after the function returns.
  3. Spawn a separate R process and call exec( R CMD BATCH batch_file ) to perform the function, and record that process’s process ID (PID).
4. Show a modal window over the dialog with an endlessly repeating progress bar, and a Cancel button. If the Cancel button is pressed, it will kill the spawned R process using taskkill PID and close the progress window.

5. Monitor periodically for the saved session file to appear in the working directory, and then load the session file and close the progress window.

**Dialog markups that work for all items**

- `arg_name.dialogItem = arg_value` sets the dialog item value, OR,
- `arg_name.dialogItem = substitute(expression)` sets the value of the widget to the evaluated expression.

- `label = "This widget specifies x"` sets the label for the widget frame.
- `tooltip = "the tooltip string"` sets a tooltip over the dialog item
- `BREAK = TRUE` stops adding widgets to the end of the column and starts a new column.
- `indent= 10` adds a 10 pixel indentation to the left hand side of the widget frame which can be useful to show hierarchical organization. `signal = c(signal_type, signal_function, widget1, widget2, ..., user_data=NULL)` specifies a widget signal (see below). More than one signal can be set for each widget.

- `signal.at.startup = FALSE` will NOT send the "default" signal from this dialog element when the dialog is initialized.

- `set.sensitive=FALSE` will gray out the corresponding widget

- `suppress=TRUE` will prevent the widget's value from being passed to the function. This is useful for "dummy" widgets that contain lists of values to choose from.

- `visible=FALSE` will just not display the widget. Its value will still be passed to the function and it can be called normally by signaling functions.

**Dialog Items**

This is a list of the currently available widgets that can be specified using markup, together with additional markup that can be used to specify their behavior.

As well as that, there are `set.value()` and `get.value()` methods that work on most widgets, and also signals which they can emit (see Signal Handling). `set.value()` takes an optional `propagate` argument which is `TRUE` by default. If this is set to `FALSE`, then the widget won’t emit its signal.

This list specifies what the `set.value` and `get.value` methods return for each widget and what behavior leads to each widget emitting a signal.
stringItem: An entry box which can take text input.

```r
tax.stringItem = "hi there"
```

Set and Get Calls:

- `get.value(item)` returns the string value.
- Signals:
  - "default" is emitted when Enter is pressed.

numericItem: A stringItem which coerces the value to a number.

```r
x.numericItem = 1.3
```

Set and Get Calls:

- `get.value(item)` returns the numeric value.

rangeItem: A horizontal slider for returning a numeric value.

```r
x.rangeItem = c(value=1, from=0, to=2, by=0.1)
```

The value sets the value, min, max and step.

Set and Get Calls:

- `get.value(item)` returns the numeric value.
- Signals:
  - "default" is emitted when the range is altered.

integerItem: A spin button which takes an integer value.

```r
z.integerItem = 1

z.integerItem = c(value=1, from=0, to=10, by=1)
```

Set and Get Calls:

- `get.value(item)` returns the integer value.
  - The value can be specified in two ways, first using a single number,
  - second with a named vector containing the value and the min, max and step of the spin button.

trueFalseItem: A check box.

```r
an.option.trueFalseItem = TRUE
```

This widget will not be put in a frame but rather have its label set to any label markup.

Set and Get Calls:

- `get.value(item)` returns the check box state.
- `set.value(item, value, propagate=TRUE)` sets the item to the logical value.
- Signals:
  - "default" is emitted when the check box is changed.

choiceItem: A combo box allowing selection of a value.

```r
x.choiceItem = c(choice1, value=choice2, choice3, ...)
```

where "value=" sets the initial choice.

Additional markup:

- `by.index=T` will return the selected indices rather than their strings.
- `item.labels = c("A", "B", "C")` will display the choices as these labels rather than the values given.

The `item.labels` vector must be the same length as the values vector if it exists.

Set and Get Calls:

- `get.value(item)` returns the selected choice.
- `get.value(item, selected=F)` returns all of the choices in the choiceItem.
set.value(item, value, propagate=TRUE) sets the available choices in the choiceItem to the vector (or possibly NULL) value.
Signals:
"default" is emitted when the selection is changed. Note that popping up the list box will not signal.

radiobuttonItem: Identically specified to choiceItem but shows radio buttons rather than a combo box. Better for small numbers of choices or longer choice names. Also, the choices in radiobuttonItem cannot be changed using set.value(), unlike choiceItem.
Markup for radiobuttonItem is the same as choiceItem.
Set and Get Calls:
get.value(item) returns the selected choice.
set.value(item, value, propagate=TRUE) sets the item to the value and throws an error if it is not one of the available choices.
Signals:
"default" is emitted when the selection is changed.

fileItem: a label and a button. When the button is pressed a file name can be selected or typed in.
fileName.fileItem = "C:/R/test.R"
Additional markup:
extension = "xls" opens *.xls
type = "open", "save", "selectdir" gives options to open, save or select directories
multiple = TRUE allows multiple file selection (not compatible with type = "selectdir")
Set and Get Calls:
get.value(item) returns a string or vector of strings corresponding to the selected files or directories. No set.value() call yet.
Signals:
"default" is emitted when any file is selected.

objectItem: a label and a button. When the button is pressed a chooser dialog opens allowing selection of objects from the global environment (using a gWidgets gvarbrowser) on either double click or select and close.
Its value is a string containing the name of the default object, otherwise NULL or "".
data.objectItem = "iris"
Additional markup:
data.types ="list" specifies a list of data types. If your selected object from the chooser doesn’t match that type, it won’t change the selected value.
as.character =TRUE means that the name of the selected dataset rather than the symbol will get passed back to the function.
Set and Get Calls:
get.value(item) returns the selected object name(the string) rather than the object itself. This is slightly inconsistent with objectItem’s default returned value to the function which is the object symbol. No set.value() call yet.
Signals:
"default" is emitted when any object is selected.

dataframeItem: identical to objectItem with default data.type markup of c("data.frame", "matrix").
Also, if a gWidgetsRGtk2 gvarbrowser is specified in run.dialog(..., var.browser), the FIRST dataframeItem in the dialog will have its value set to any dataframe or array selected in the gvarbrowser.
variableSelectorItem: a widget which displays a list with checkboxes to choose a subset of strings. The check box column header is labeled "Select All" and clicking it will alternate between selecting and deselecting all items.

```r
myColumns$variableSelectorItem = letters[1:5]
myColumns$variableSelectorItem = substitute(colnames(iris))
```

A variableSelectorItem will pass to the function a (possibly zero-length) vector containing the selected strings.

Set and Get Calls:
- `get.value(item)` returns the selected values.
- `set.value(item, letters[1:5], propagate=TRUE)` sets the values in the item.

Signals:
- "default" is emitted when the check boxes are altered anywhere.

listItem: an widget containing a simple list box set in a panel with buttons labeled with right and left arrows next to it. The list allows multiple selections.

```r
myColumns$listItem = substitute(colnames(iris))
nothingYet$listItem = NULL
```

Additional markup:
- `show.arrows = FALSE` turns displaying the arrow buttons off.
- `max.items = 2` sets the maximum number of items the listItem can contain to 2.

Set and Get Calls:
- `get.value(item, select=TRUE)` returns the selected values only.
- `get.value(item, select=FALSE)` returns all values.
- `set.value(item, letters[1:5], propagate=TRUE)` sets the values in the list item.

Signals:
- "default" is emitted when the list is clicked anywhere.
- "add" is emitted when the right-arrow button is clicked.
- "subtract" is emitted when the left-arrow button is clicked.

buttonItem: a frameless button.

```r
button$buttonItem = "Click Me"
```

Set and Get Calls:
- None, yet.

Signals:
- "clicked" is emitted when the button is clicked.

**Dot Arguments**

If a markup item called ....stringItem is present, run.dialog() will display it as a stringItem widget with an "Additional Arguments" label.

When the dialog "OK" is clicked run.dialog() will run `eval(parse(text=get.value(text)))` on whatever text is put in that markup, then place any additional items as additional arguments to the function.

**Signaling** Signal handling can be specified to support some commonly required function dialog behaviors. For example, deselecting a checkbox might set another widget to insensitive (grayed out), or some items might be moved between list boxes.
Signaling is specified using the signaling markup, set after a dialog item: \texttt{signal = c\texttt{\{signal\_type, signal\_function, widget1\_arg, widget2\_arg, ...\}} containing the following specifications:

\texttt{signal\_type} is a string, for example "default", "clicked", etc.

All the dialogItems will broadcast a signal called "default" whenever something happens to them corresponding to a user interaction.

This is usually what you're going to want to connect a signal to, but there are exceptions.

\texttt{signal\_function} is a function or a name of a function visible in the environment \texttt{run.dialog} is called from. It has this signature:

\texttt{signal\_function(item, widget1, widget2, ..., user\_data=NULL)}

item is the widget that sent the signal;

widget1, widget2 are GtkWidget objects specified by;

user\_data is the optionally specified user data

\texttt{widget1\_arg, widget2\_arg, ...} are the function argument strings corresponding to the widgets being passed to the \texttt{signal\_function}.

user\_data is optional additional data to send to the function.

Within your \texttt{signal\_function}, you're going to want to be able to get and set the values your widgets take.

You can do this using the \texttt{get\_value(widget)} and \texttt{set\_value(widget)} calls. In some cases these calls can take additional arguments. See the DensityHistogram example below.

Actually, we could use the built-in function \texttt{toggle\_sensitive} which does the same thing (the commented-out \texttt{#signal} line). In this case, passing a string as the signaling function will make \texttt{run.dialog} search for a function with this name.

One further question is, how does \texttt{run.dialog} set the rangeItem to be insensitive the first time it opens? By default, \texttt{run.dialog()} calls the "default" signal from every one of the widgets when it opens, unless either

(a) they are set to being insensitive or
(b) the widget \texttt{signal.on.startup} markup is set to \texttt{FALSE}.

Because the default signal from some widgets can set others to being insensitive, this allows reasonable behavior for hierarchically grouped toggled sensitive widgets. On the other hand, it may still be desirable to prevent widgets firing their signals when the dialog is starting, for example if you have many widgets performing some action like refreshing a list. In this case, setting \texttt{signal.on.startup} to \texttt{FALSE} for most of them should fix that problem.
Examples

## Not run:

```r
# A simple histogram plotting example:
SimpleHistogram = function(N) hist(rnorm(N))

SimpleHistogram.dialog = list(label = "A density-plotting histogram dialog",
N.integerItem = 50, label = "Value of N")

run.dialog(SimpleHistogram)
```

## example 2

```r
# Function to plot a histogram, optionally adding a density plot with
# adjustable bandwidth. If add.density is false, gray out the bandwidth range.

# This is what's going on in the "signal" markup from the add.density widget:
# We specify the "default" signal from the add.density trueFalseItem
# Then specify the signal function, which takes the add.density
# trueFalseItem as its first argument and the bw.rangeItem
# as its second. In the function, we first of all
# get the value of the trueFalseItem (which is TRUE or FALSE)
# and call gtkWidgetSetSensitive on bw.widget with this value
# (see the Gtk Help for gtkWidget)
# The last element of the signal markup is the string "bw"
# which run.dialog() replaces with the corresponding bw.rangeItem and
# passes to the called signal function as its second argument.

DensityHistogram = function(N, add.density, bw=NULL) {
  x <- rnorm(N)
  hist(x, probability=add.density)
  if(add.density) points(density(x, bw=bw), type = "l")
}

DensityHistogram.dialog = list(
  #keep.open = TRUE, # uncomment to keep the dialog open
  label = "A density-plotting histogram dialog",
  N.integerItem = c(value=100, from=10, to=1000, by=10), label = "Value of X",
  add.density.trueFalseItem = FALSE, label = "Add Density Plot?",
  signal = c("default",
      function(item, bw.widget) bw.widget$setSensitive(get.value(item)), "bw"),
  # signal = c("default", "toggle.sensitive", "bw"), # using the built-in
  bw.rangeItem = c(value=0.3, from=0, to=2, by=0.05), label = "Bandwidth")

run.dialog(DensityHistogram)
```

# end example 2

```r
# Angle display demo, from Graham Williams
demo(MakeAngle)
```
# Cancel dialog demo
demo(ProgressBar)

## End\(\text{Not run}\)

### safe.eval

**Description**

Equivalent to \(\text{get(x)}\) for \(x\) of the form (i) "object" (ii) "list\$object"

**Usage**

\[
\text{safe.eval(text, envir=parent.env(environment()))}
\]

**Arguments**

- **text**: text to eval
- **envir**: env to evaluate in

**Details**

Evaluates \(\text{get(x)}\) for \(x\) of the form (i) "object" (ii) "list\$object"

### set.value

**Description**

generic value setting method

**Usage**

\[
\text{set.value(dlg.item, ...)}
\]

**Arguments**

- **dlg.item**: gtkWidget
- **...**: args
Details

Complementary to `get.value()`, this is the generic function for setting the value of a widget. See `?run.dialog` for details. `set.value` comes with an optional `propagate` argument which is TRUE by default. If it's set to FALSE, then calling `set.value(widget, value, propagate=FALSE)` will not trigger the widget's default signal if this is bound, which can be useful for more complex behaviors.

Description

S3 data extraction method

Usage

```r
## S3 method for class 'GtkDfEdit'
x[i, j, drop=TRUE]
```

Arguments

- `x`: The RGtk2DfEdit object
- `i`: Row index
- `j`: Column index
- `drop`: passed to extraction for data frame

Details

Grabs data frame then passes onto `.data.frame` method

Value

The extracted entries
Index

*Topic interface
  get.value, 4
  get_all_tables, 5
  GetTaskPasteIn, 4
  gtkDFEditCommandData, 10
  gtkDFEditGetDatasetName, 14
  gtkDFEditSetDatasetName, 19
  my_choose_files, 21
  object.exists, 21
  quick_message, 22
  run.dialog, 22
  safe.eval, 32
  set.value, 32
[.GtkDFEdit, 33
[.gtkDFEdit (gtkDFEdit), 5
[<-gtkDFEdit (gtkDFEdit), 5

dfedit, 2, 10
dfview, 3

gtkDFEdit, 5, 13, 20