Package 'editbl'

January 31, 2025

Type Package

Version 1.1.0

Date 2025-01-31

Title 'DT' Extension for CRUD (Create, Read, Update, Delete) Applications in 'shiny'

Maintainer Jasper Schelfhout <jasper.schelfhout@openanalytics.eu>

Description The core of this package is a function eDT() which enhances DT::datatable() such that it can be used to interactively modify data in 'shiny'. By the use of generic 'dplyr' methods it supports many types of data storage, with relational databases ('dbplyr') being the main use case.

License GPL-3

Imports shiny, shinyjs, DT, tibble, dplyr, rlang, uuid, fontawesome (>= 0.4.0)

Suggests testthat, dtplyr, data.table, vctrs, RSQLite, dbplyr, glue, DBI, bit64, knitr, dm

URL https://github.com/openanalytics/editbl

BugReports https://github.com/openanalytics/editbl/issues

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Jasper Schelfhout [aut, cre], Maxim Nazarov [rev], Daan Seynaeve [rev], Lennart Tuijnder [rev]

Repository CRAN

Date/Publication 2025-01-31 10:40:02 UTC

Contents

`addButtons` *Add modification buttons as a column*

Description

Add modification buttons as a column

Usage

```
addButtons(
  df,
  columnName,
  ns,
  iCol = "i",
  canEditRow = TRUE,
  canDeleteRow = TRUE,
  statusCol = "status"
)
```

Arguments

df	data.frame
columnName	character(1)
ns	namespace function
iCol	character(1) name of column containing a unique identifier.
canEditRow	can be either of the following: • logical, e.g. TRUE or FALSE • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.
canDeleteRow	can be either of the following: • logical, e.g. TRUE or FALSE • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.
statusCol	character(1) name of column with general status (e.g. modified or not). if NULL, the data is interpreted as 'unmodified'.

Value

df with extra column containing buttons

Author(s)

Jasper Schelfhout

---

beginTransaction *Start a transaction for a tibble*

Description

Start a transaction for a tibble

Usage

```
beginTransaction(tbl)
```

Arguments

tbl	tbl

Author(s)

Jasper Schelfhout

canXXXRowTemplate *Re-usable documentation*

Description

Re-usable documentation

Usage

```
canXXXRowTemplate(canEditRow, canDeleteRow)
```

Arguments

canEditRow	can be either of the following: • logical, e.g. TRUE or FALSE • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.
canDeleteRow	can be either of the following: • logical, e.g. TRUE or FALSE • function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.

castForDisplay *Cast columns in* data.frame *to editable types in datatable*

Description

Cast columns in data.frame to editable types in datatable

Usage

```
castForDisplay(data, cols = colnames(data))
```

Arguments

data	data.frame
cols	character columns to perform casting on.

Value

data.frame with some columns cast to another type

Author(s)

Jasper Schelfhout

castFromTbl *Cast tbl to class of template*

Description

Cast tbl to class of template

Usage

```
castFromTbl(tbl, template)
```

Arguments

tbl	tbl
template	tabular object like data.frame or data.table or tbl.

Value

tbl cast to the type of template

Author(s)

Jasper Schelfhout

castToFactor *Cast all columns that exist in a foreignTbl to factor*

Description

Cast all columns that exist in a foreignTbl to factor

Usage

```
castToFactor(data, foreignTbls)
```

Arguments

data	data.frame
foreignTbls	list of foreign tbls as created by foreignTbl

Details

Can be used to fixate possible options when editing.

Value

data.frame

Author(s)

Jasper Schelfhout

---

castToSQLSupportedType *Cast the data type to something supported by SQL.*

Description

Cast the data type to something supported by SQL.

Usage

```
castToSQLSupportedType(x)
```

Arguments

x	single value or vector of values

Value

x, possibly cast to different type

Author(s)

Jasper Schelfhout

---

castToTbl *Cast data to tbl*

Description

Cast data to tbl

Usage

```
castToTbl(data)
```

Arguments

data	object

Value

tbl

Author(s)

Jasper Schelfhout

`castToTemplate` *Cast* `tbl` *or* `data.frame` *x to the types of the template*

Description

Cast `tbl` or `data.frame` x to the types of the template

Usage

```
castToTemplate(x, template)
```

Arguments

`x`	`data.frame`, `tbl` or `data.table`
`template`	`data.frame`, `tbl` or `data.table`

Details

If template is a `tbl` with database source, convert to an in-memory tibble with same data types instead.

Rownames might differ or get lost.

Value

object containing data of x in the class and structure of the template.

Author(s)

Jasper Schelfhout

checkForeignTbls *Check if all rows in tbl fufill* foreignTbl *constraints*

Description

Check if all rows in tbl fufill foreignTbl constraints

Usage

```
checkForeignTbls(tbl, foreignTbls)
```

Arguments

tbl	tbl
foreignTbls	list of foreign tbls as created by foreignTbl

Value

logical stating if tbl fufills all constraints imposed by all foreign tbls.

Author(s)

Jasper Schelfhout

coalesce *Return first non* NULL *argument*

Description

Return first non NULL argument

Usage

```
coalesce(...)
```

Arguments

...	set of arguments

Author(s)

Jasper Schelfhout

coerceColumns *Cast columns to the type of the template*

Description

Cast columns to the type of the template

Usage

```
coerceColumns(template, x)
```

Arguments

template	data.frame
x	data.frame

Details

only affects columns in both the template and x

coerceValue DT::coerceValue *with better* POSIXct *support*

Description

DT::coerceValue with better POSIXct support

Usage

```
coerceValue(val, old)
```

Arguments

val	A character string.
old	An old value, whose type is the target type of val.

Details

Will assume UTC in case no timezone is specified.

Author(s)

Jasper Schelfhout

commitTransaction *Start a transaction for a tibble*

Description

Start a transaction for a tibble

Usage

```
commitTransaction(tbl)
```

Arguments

tbl	tbl

Author(s)

Jasper Schelfhout

connectDB *Connect to a database.*

Description

Connect to a database.

Usage

```
connectDB(
  dbname = system.file("extdata", "chinook.sqlite", package = utils::packageName()),
  drv = RSQLite::SQLite(),
  ...
)
```

Arguments

dbname	character(0)
drv	database driver
...	arguments passed to DBI::dbConnect

Details

Connects by default to a test SQLite database originally obtained here: chinook_git

Value

database connection

Examples

```
conn <- connectDB()
DBI::dbDisconnect(conn)
```

`createButtons` *Create buttons to modify the row.*

Description

Create buttons to modify the row.

Usage

```
createButtons(
  row,
  suffix,
  ns,
  canEditRow = TRUE,
  canDeleteRow = TRUE,
  statusCol = "status"
)
```

Arguments

`row`	`tibble` with single row
`suffix`	`character(1)`
`ns`	`character(1)` namespace
`canEditRow`	can be either of the following: • `logical`, e.g. TRUE or FALSE • `function`. Needs as input an argument `row` which accepts a single row `tibble` and as output TRUE/FALSE.
`canDeleteRow`	can be either of the following: • `logical`, e.g. TRUE or FALSE • `function`. Needs as input an argument `row` which accepts a single row `tibble` and as output TRUE/FALSE.
`statusCol`	`character(1)` name of column with general status (e.g. modified or not). if `NULL`, the data is interpreted as 'unmodified'.

Details

buttons used per row in the app.

Value

character(1) HTML

---

createDeleteButtonHTML *Generate HTML for an in-row delete button*

Description

Generate HTML for an in-row delete button

Usage

```
createDeleteButtonHTML(ns = "%1$s", suffix = "%2$s", disabled = FALSE)
```

Arguments

ns	character(1) namespace
suffix	character(1) id of the row
disabled	logical(1) wether or not the button has to be disabled

Value

character(1) HTML

---

createDeleteButtonHTML_shiny *Helper function to write HTML*

Description

Helper function to write HTML

Usage

```
createDeleteButtonHTML_shiny(ns = "%1$s", suffix = "%2$s", disabled = FALSE)
```

Arguments

ns	character(1) namespace
suffix	character(1) id of the row
disabled	logical(1) wether or not the button has to be disabled

Details

only to be used interactively. sprintf() implementation is faster.

See Also

createEditButtonHTML

---

createEditButtonHTML *Generate HTML for an in-row edit button*

Description

Generate HTML for an in-row edit button

Usage

```
createEditButtonHTML(ns, suffix, disabled = FALSE)
```

Arguments

ns	character(1) namespace
suffix	character(1) id of the row
disabled	logical(1) wether or not the button has to be disabled

Value

character(1) HTML

---

createEditButtonHTML_shiny *Helper function to write HTML*

Description

Helper function to write HTML

Usage

```
createEditButtonHTML_shiny(ns = "%1$s", suffix = "%2$s", disabled = FALSE)
```

Arguments

ns	character(1) namespace
suffix	character(1) id of the row
disabled	logical(1) wether or not the button has to be disabled

Details

only to be used interactively. sprintf() implementation is faster.

See Also

createEditButtonHTML

`customButton` *Generate a custom button for* `eDT`

Description

Generate a custom button for `eDT`

Usage

```
customButton(id, label, icon = "", disabled = FALSE)
```

Arguments

`id`	`character(1)`, namespaced id
`label`	`character(1)`
`icon`	`shiny::icon`
`disabled`	`logical`. Whether or not the button should start in a disabled state.

Details

Combines elements of `shiny::actionButton` and datatable options

Value

list to be used in `eDT(options = list(buttons = xxx))`

Author(s)

Jasper Schelfhout

Examples

```
if(interactive()){

  ui <- eDTOutput("data")
  server <- function(input,output,session){
    b <- customButton('print', label = 'print')
    eDT_result <- eDT(id = "data", mtcars, options = list(buttons = list("save", b)))
    observeEvent(input$print,{
          print(eDT_result$state())
    })
```

```
  }
  shinyApp(ui,server)
}
```

`demoServer_custom` *Server of the mtcars demo app*

Description

Server of the mtcars demo app

Usage

```
demoServer_custom(id, x)
```

Arguments

`id`	`character(1)`
`x`	`tbl`

Value

NULL, just executes the module server.

Author(s)

Jasper Schelfhout

`demoServer_DB` *Server of the DB demo app*

Description

Server of the DB demo app

Usage

```
demoServer_DB(id, conn)
```

Arguments

`id`	`character(1)`
`conn`	database connection object as given by `dbConnect`.

Value

NULL, just executes the module server.

Author(s)

Jasper Schelfhout

---

demoServer_mtcars *Server of the mtcars demo app*

Description

Server of the mtcars demo app

Usage

```
demoServer_mtcars(id)
```

Arguments

id	character(1)

Value

NULL, just executes the module server.

Author(s)

Jasper Schelfhout

---

demoUI_custom *UI of the demo mtcars app*

Description

UI of the demo mtcars app

Usage

```
demoUI_custom(id)
```

Arguments

id	character(1)

Value

HTML

Author(s)

Jasper Schelfhout

`demoUI_DB` *UI of the DB demo app*

Description

UI of the DB demo app

Usage

```
demoUI_DB(id, conn)
```

Arguments

`id`	`character(1)`
`conn`	database connection object as given by `dbConnect`.

Value

HTML

Author(s)

Jasper Schelfhout

`demoUI_mtcars` *UI of the demo mtcars app*

Description

UI of the demo mtcars app

Usage

```
demoUI_mtcars(id)
```

Arguments

`id`	`character(1)`

Value

HTML

Author(s)

Jasper Schelfhout

`devServer` *Server of the development app*

Description

Server of the development app

Usage

```
devServer(id, conn)
```

Arguments

`id`	`character(1)`
`conn`	database connection object as given by `dbConnect`.

Value

NULL, just executes the module server.

Author(s)

Jasper Schelfhout

`devUI` *UI of the development app*

Description

UI of the development app

Usage

```
devUI(id, conn)
```

Arguments

`id`	`character(1)`
`conn`	database connection object as given by `dbConnect`.

Value

HTML

Author(s)

Jasper Schelfhout

`disableDoubleClickButtonCss` *Function to generate CSS to disable clicking events on a column*

Description

Function to generate CSS to disable clicking events on a column

Usage

```
disableDoubleClickButtonCss(id)
```

Arguments

`id`	`character(1)` namespaced id of the datatable

Details

https://stackoverflow.com/questions/60406027/how-to-disable-double-click-reactivity-for-specific-c

https://stackoverflow.com/questions/75406546/apply-css-styling-to-a-single-dt-datatable

Value

`character` CSS

`eDT` *Create a modifieable datatable.*

Description

Create a modifieable datatable.

Usage

```
eDT(
  data,
  options = list(dom = "Bfrtlip", keys = TRUE, ordering = FALSE, autoFill = list(update =
    FALSE, focus = "focus"), buttons = list("add", "undo", "redo", "save")),
  class = "display",
  callback = NULL,
  rownames = FALSE,
  colnames = NULL,
  container,
  caption = NULL,
  filter = c("none", "bottom", "top"),
  escape = TRUE,
  style = "auto",
  width = NULL,
  height = NULL,
  elementId = NULL,
  fillContainer = getOption("DT.fillContainer", NULL),
  autoHideNavigation = getOption("DT.autoHideNavigation", NULL),
  selection = "none",
  extensions = c("KeyTable", "AutoFill", "Buttons"),
  plugins = NULL,
  editable = list(target = "cell"),
  id,
  keys = NULL,
  in_place = FALSE,
  format = function(x) {
    x
  },
  foreignTbls = list(),
  statusColor = c(insert = "#e6e6e6", update = "#32a6d3", delete = "#e52323"),
  inputUI = editbl::inputUI,
  defaults = tibble(),
  env = environment(),
  canEditRow = TRUE,
  canDeleteRow = TRUE,
  utilityColumns = NULL
)
```

Arguments

data	tbl. The function will automatically cast to tbl if needed.
options	a list of initialization options (see https://datatables.net/reference/option/); the character options wrapped in JS() will be treated as literal JavaScript code instead of normal character strings; you can also set options globally via options(DT.options = list(...)), and global options will be merged into this options argument if set

class	the CSS class(es) of the table; see https://datatables.net/manual/styling/classes
callback	the body of a JavaScript callback function with the argument table to be applied to the DataTables instance (i.e. table)
rownames	TRUE (show row names) or FALSE (hide row names) or a character vector of row names; by default, the row names are displayed in the first column of the table if exist (not NULL)
colnames	if missing, the column names of the data; otherwise it can be an unnamed character vector of names you want to show in the table header instead of the default data column names; alternatively, you can provide a *named* numeric or character vector of the form 'newName1' = i1, 'newName2' = i2 or c('newName1' = 'oldName1', 'newName2' = 'oldName2', ...), where newName is the new name you want to show in the table, and i or oldName is the index of the current column name
container	a sketch of the HTML table to be filled with data cells; by default, it is generated from htmltools::tags$table() with a table header consisting of the column names of the data
caption	the table caption; a character vector or a tag object generated from htmltools::tags$caption()
filter	whether/where to use column filters; none: no filters; bottom/top: put column filters at the bottom/top of the table; range sliders are used to filter numeric/date/time columns, select lists are used for factor columns, and text input boxes are used for character columns; if you want more control over the styles of filters, you can provide a named list to this argument; see Details for more
escape	whether to escape HTML entities in the table: TRUE means to escape the whole table, and FALSE means not to escape it; alternatively, you can specify numeric column indices or column names to indicate which columns to escape, e.g. 1:5 (the first 5 columns), c(1, 3, 4), or c(-1, -3) (all columns except the first and third), or c('Species', 'Sepal.Length'); since the row names take the first column to display, you should add the numeric column indices by one when using rownames
style	either 'auto', 'default', 'bootstrap', or 'bootstrap4'. If 'auto', and a **bslib** theme is currently active, then bootstrap styling is used in a way that "just works" for the active theme. Otherwise, DataTables 'default' styling is used. If set explicitly to 'bootstrap' or 'bootstrap4', one must take care to ensure Bootstrap's HTML dependencies (as well as Bootswatch themes, if desired) are included on the page. Note, when set explicitly, it's the user's responsibility to ensure that only one unique 'style' value is used on the same page, if multiple DT tables exist, as different styling resources may conflict with each other.
width, height	Width/Height in pixels (optional, defaults to automatic sizing)
elementId	An id for the widget (a random string by default).
fillContainer	TRUE to configure the table to automatically fill it's containing element. If the table can't fit fully into it's container then vertical and/or horizontal scrolling of the table cells will occur.

autoHideNavigation
: TRUE to automatically hide navigational UI (only display the table body) when the number of total records is less than the page size. Note, it only works on the client-side processing mode and the 'pageLength' option should be provided explicitly.

selection
: the row/column selection mode (single or multiple selection or disable selection) when a table widget is rendered in a Shiny app; alternatively, you can use a list of the form `list(mode = 'multiple',selected = c(1, 3, 8), target = 'row', selectable = c(-2, -3))` to pre-select rows and control the selectable range; the element `target` in the list can be `'column'` to enable column selection, or `'row+column'` to make it possible to select both rows and columns (click on the footer to select columns), or `'cell'` to select cells. See details section for more info.

extensions
: a character vector of the names of the DataTables extensions (https://datatables.net/extensions/index)

plugins
: a character vector of the names of DataTables plug-ins (https://rstudio.github.io/DT/plugins.html). Note that only those plugins supported by the DT package can be used here. You can see the available plugins by calling `DT:::available_plugins()`

editable
: FALSE to disable the table editor, or TRUE (or `"cell"`) to enable editing a single cell. Alternatively, you can set it to `"row"` to be able to edit a row, or `"column"` to edit a column, or `"all"` to edit all cells on the current page of the table. In all modes, start editing by doubleclicking on a cell. This argument can also be a list of the form `list(target = TARGET, disable = list(columns = INDICES))`, where TARGET can be `"cell"`, `"row"`, `"column"`, or `"all"`, and INDICES is an integer vector of column indices. Use the list form if you want to disable editing certain columns. You can also restrict the editing to accept only numbers by setting this argument to a list of the form `list(target = TARGET, numeric = INDICES)` where INDICES can be the vector of the indices of the columns for which you want to restrict the editing to numbers or `"all"` to restrict the editing to numbers for all columns. If you don't set `numeric`, then the editing is restricted to numbers for all numeric columns; set `numeric = "none"` to disable this behavior. It is also possible to edit the cells in text areas, which are useful for large contents. For that, set the `editable` argument to a list of the form `list(target = TARGET, area = INDICES)` where INDICES can be the vector of the indices of the columns for which you want the text areas, or `"all"` if you want the text areas for all columns. Of course, you can request the numeric editing for some columns and the text areas for some other columns by setting `editable` to a list of the form `list(target = TARGET, numeric = INDICES1, area = INDICES2)`. Finally, you can edit date cells with a calendar with `list(target = TARGET, date = INDICES)`; the target columns must have the `Date` type. If you don't set `date` in the `editable` list, the editing with the calendar is automatically set for all `Date` columns.

id
: `character(1)` module id

keys
: `character`. Defaults to all columns under the assumption that at least every row is unique.

in_place — logical. Whether to modify the data object in place or to return a modified copy.

format — function accepting and returning a datatable

foreignTbls — list. List of objects created by foreignTbl

statusColor — named character. Colors to indicate status of the row.

inputUI — function. UI function of a shiny module with at least arguments id data and .... #' elements with inputIds identical to one of the column names are used to update the data.

defaults — expression that evaluates to a tibble with (a subset of) columns of the data. It will be evaluated for each new row in the environment defined by 'env'. This allows for defaults like Sys.time() or uuid::UUIDgenerate() as well as dynamic inputs.

env — environment in which the server function is running. Should normally not be modified.

canEditRow — can be either of the following:

- logical, e.g. TRUE or FALSE
- function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.

canDeleteRow — can be either of the following:

- logical, e.g. TRUE or FALSE
- function. Needs as input an argument row which accepts a single row tibble and as output TRUE/FALSE.

utilityColumns — named character vector. Defines names for (hidden) utility columns used by eDT to keep track of modifications. Should normally only be adjusted in rare case of name clashes with data.

```
c(
  status = '_editbl_status',
  buttons = '_editbl_buttons',
  identity = '_editbl_identity',
  deleted = '_editbl_deleted'
 )
```

Details

Works the same as datatable. This function is however a shiny module and comes with additional arguments and different defaults. Instead of having output$id = renderDT(DT::datatable(iris)), eDT(id = 'id', data = iris) should be used on the server side. On the UI side eDTOutput should be used instead of DTOutput.

Can also be used as standalone app when not ran in reactive context.

All arguments except 'id' and 'env' can be normal objects or reactive objects.

Value

list

- result `reactive` modified version of `data` (saved)
- state `reactive` current state of the `data` (unsaved)
- selected `reactive` selected rows of the `data` (unsaved)

Author(s)

Jasper Schelfhout

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  # tibble support
  modifiedData <- editbl::eDT(tibble::as_tibble(mtcars))

  # data.table support
  modifiedData <- editbl::eDT(dtplyr::lazy_dt(data.table::data.table(mtcars)))

  # database support
  tmpFile <- tempfile(fileext = ".sqlite")
  file.copy(system.file("extdata", "chinook.sqlite", package = 'editbl'), tmpFile)

  conn <- editbl::connectDB(dbname = tmpFile)
  modifiedData <- editbl::eDT(dplyr::tbl(conn, "Artist"), in_place = TRUE)
  DBI::dbDisconnect(conn)

  unlink(tmpFile)

  # Within shiny
  library(shiny)
  library(editbl)
  shinyApp(
    ui = fluidPage(fluidRow(column(12, eDTOutput('tbl')))),
    server = function(input, output) {
      eDT('tbl',iris,)
    }
  )

  # Custom inputUI
  editbl::eDT(mtcars, inputUI = function(id, data){
    ns <- NS(id)
    textInput(
    ns("mpg"),
    label = "mpg",
    value = data$mpg)})

  # Do not allow delete
  editbl::eDT(mtcars, canDeleteRow = FALSE)
```

```
}
```

`eDTOutput` *UI part of* `eDT`

Description

UI part of `eDT`

Usage

```
eDTOutput(id, ...)
```

Arguments

`id`	`character(1)`
`...`	arguments passed to `DTOutput`

Details

Works exactly like `DTOutput` apart from the fact that instead of the `outputId` argument, `id` is requested. Reason being that this function is a UI to a shiny module. This means that the datatable can be found under the id `'{namespace}-{id}-DT'` instead of `'{namespace}-{outputId}'`.

Also some minor CSS and javascript is executed for functional puposes.

Value

HTML

Author(s)

Jasper Schelfhout

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  # tibble support
  modifiedData <- editbl::eDT(tibble::as_tibble(mtcars))

  # data.table support
  modifiedData <- editbl::eDT(dtplyr::lazy_dt(data.table::data.table(mtcars)))

  # database support
  tmpFile <- tempfile(fileext = ".sqlite")
  file.copy(system.file("extdata", "chinook.sqlite", package = 'editbl'), tmpFile)

  conn <- editbl::connectDB(dbname = tmpFile)
```

```
  modifiedData <- editbl::eDT(dplyr::tbl(conn, "Artist"), in_place = TRUE)
  DBI::dbDisconnect(conn)

  unlink(tmpFile)

  # Within shiny
  library(shiny)
  library(editbl)
  shinyApp(
    ui = fluidPage(fluidRow(column(12, eDTOutput('tbl')))),
    server = function(input, output) {
      eDT('tbl',iris,)
    }
  )

  # Custom inputUI
  editbl::eDT(mtcars, inputUI = function(id, data){
    ns <- NS(id)
    textInput(
    ns("mpg"),
    label = "mpg",
    value = data$mpg)})

  # Do not allow delete
  editbl::eDT(mtcars, canDeleteRow = FALSE)
}
```

---

eDT_app *Open interactive app to explore and modify data*

Description

Open interactive app to explore and modify data

Usage

```
eDT_app(...)
```

Arguments

...	arguments past to eDT

Details

When eDT is not used within the server of a shiny app, it will call this function to start up a shiny app itself. Just as DT::datatable() displays a table in the browser when called upon interactively.

Value

data (or a modified version thereof) once you click 'close'

eDT_app_server *Server of eDT_app*

Description

Server of eDT_app

Usage

```
eDT_app_server(moduleId = "nevergonnagiveyouup", ...)
```

Arguments

moduleId	character(1) id to connect with eDT_app_server
...	arguments passed to eDT

Value

moduleServer which on application stop returns version of x with made changes

Author(s)

Jasper Schelfhout

eDT_app_ui *UI of eDT_app*

Description

UI of eDT_app

Usage

```
eDT_app_ui(moduleId = "nevergonnagiveyouup", eDTId = "nevergonnaletyoudown")
```

Arguments

moduleId	character(1) id to connect with eDT_app_server
eDTId	character(1) id to connect eDTOutput to eDT within the module.

Value

HTML

Author(s)

Jasper Schelfhout

evalCanDeleteRow *Determine if a row can be deleted*

Description

Determine if a row can be deleted

Usage

```
evalCanDeleteRow(row, canDeleteRow = TRUE, statusCol = "status")
```

Arguments

row	tibble, single row
canDeleteRow	function with argument 'row' defining logic on wether or not the row can be modified. Can also be logical TRUE or FALSE.
statusCol	character(1) name of column with general status (e.g. modified or not).

Details

calling this around the user passed on function ensures that newly inserted rows are being excempt from the logic. Moreover, the output of the function can be checked.

Value

boolean

Author(s)

Jasper Schelfhout

evalCanEditRow *Determine if a row can be edited*

Description

Determine if a row can be edited

Usage

```
evalCanEditRow(row, canEditRow = TRUE, statusCol = "status")
```

Arguments

row	tibble, single row.
canEditRow	function with argument 'row' defining logic on wether or not the row can be modified. Can also be logical TRUE or FALSE.
statusCol	character(1) name of column with general status (e.g. modified or not).

Details

calling this around the user passed on function ensures that newly inserted rows are being excempt from the logic. Moreover, the output of the function can be checked.

Value

boolean

Author(s)

Jasper Schelfhout

---

e_rows_insert *Insert rows into a tibble*

Description

Insert rows into a tibble

Usage

```
e_rows_insert(
  x,
  y,
  by = NULL,
  ...,
  conflict = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.

...	Other parameters passed onto methods.
conflict	For rows_insert(), how should keys in y that conflict with keys in x be handled? A conflict arises if there is a key in y that already exists in x. One of: • "error", the default, will error if there are any keys in y that conflict with keys in x. • "ignore" will ignore rows in y with keys that conflict with keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_insert. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

---

e_rows_insert.default *Insert rows into a tibble*

Description

Insert rows into a tibble

Usage

```
## Default S3 method:
e_rows_insert(
  x,
  y,
  by = NULL,
  ...,
  conflict = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
conflict	For rows_insert(), how should keys in y that conflict with keys in x be handled? A conflict arises if there is a key in y that already exists in x. One of: • "error", the default, will error if there are any keys in y that conflict with keys in x. • "ignore" will ignore rows in y with keys that conflict with keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_insert. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

`e_rows_insert.dtplyr_step` *rows_insert implementation for* `data.table` *backends.*

Description

rows_insert implementation for `data.table` backends.

Usage

```
## S3 method for class 'dtplyr_step'
e_rows_insert(x, y, by = NULL, ..., copy = FALSE, in_place = FALSE)
```

Arguments

`x, y`	A pair of data frames or data frame extensions (e.g. a tibble). `y` must have the same columns of `x` or a subset.
`by`	An unnamed character vector giving the key columns. The key columns must exist in both `x` and `y`. Keys typically uniquely identify each row, but this is only enforced for the key values of `y` when `rows_update()`, `rows_patch()`, or `rows_upsert()` are used. By default, we use the first column in `y`, since the first column is a reasonable place to put an identifier variable.
`...`	Other parameters passed onto methods.
`copy`	If `x` and `y` are not from the same data source, and `copy` is `TRUE`, then `y` will be copied into the same src as `x`. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
`in_place`	Should `x` be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When `TRUE`, a modified version of `x` is returned invisibly; when `FALSE`, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around `rows_insert`. Allows for specific implementations should the behavior differ from what's needed by `editbl`. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

---

e_rows_insert.tbl_dbi *rows_insert implementation for DBI backends.*

Description

rows_insert implementation for DBI backends.

Usage

```
## S3 method for class 'tbl_dbi'
e_rows_insert(x, y, by = NULL, ..., copy = FALSE, in_place = FALSE)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_insert. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

Examples

```
library(dplyr)

# Set up a test table
conn <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
artists_df <- data.frame(
         ArtistId = c(1,2),
         Name = c("AC/DC", "The Offspring")
)
DBI::dbWriteTable(conn, "Artist", artists_df)

# Insert new row
artists <- tbl(conn, "Artist")
DBI::dbBegin(conn)
e_rows_insert(artists,
 data.frame(ArtistId = 999, Name = "testArtist"),
 in_place = TRUE)

DBI::dbRollback(conn)
DBI::dbDisconnect(conn)
```

e_rows_update *Update rows of a tibble*

Description

Update rows of a tibble

Usage

```
e_rows_update(
  x,
  y,
  by = NULL,
  ...,
  match,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

in_place — Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables).

When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_update. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

---

e_rows_update.data.frame

rows_update implementation for data.frame backends.

Description

rows_update implementation for data.frame backends.

Usage

```
## S3 method for class 'data.frame'
e_rows_update(
  x,
  y,
  by = NULL,
  match = NULL,
  ...,
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_update. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

e_rows_update.default *Update rows of a tibble*

Description

Update rows of a tibble

Usage

```
## Default S3 method:
e_rows_update(
  x,
  y,
  by = NULL,
  ...,
  match = match,
  unmatched = c("error", "ignore"),
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
...	Other parameters passed onto methods.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

in_place — Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables).

When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_update. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

---

e_rows_update.dtplyr_step

rows_update implementation for data.table backends.

Description

rows_update implementation for data.table backends.

Usage

```
## S3 method for class 'dtplyr_step'
e_rows_update(
  x,
  y,
  by = NULL,
  match = NULL,
  ...,
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_update. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

`e_rows_update.tbl_dbi` *rows_update implementation for DBI backends.*

Description

rows_update implementation for DBI backends.

Usage

```
## S3 method for class 'tbl_dbi'
e_rows_update(
  x,
  y,
  by = NULL,
  match = NULL,
  ...,
  copy = FALSE,
  in_place = FALSE
)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.
match	named list consisting out of two equal length data.frame's with columns defined in by. This allows for updates of columns defined in by.
...	Other parameters passed onto methods.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Details

Mainly a wrapper around rows_update. Allows for specific implementations should the behavior differ from what's needed by editbl. Reason for separate method is to avoid conflicts on package loading.

Value

An object of the same type as `x`. The order of the rows and columns of `x` is preserved as much as possible. The output has the following properties:

- `rows_update()` and `rows_patch()` preserve the number of rows; `rows_insert()`, `rows_append()`, and `rows_upsert()` return all existing rows and potentially new rows; `rows_delete()` returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from `x`.
- Data frame attributes are taken from `x`.

If `in_place = TRUE`, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

Examples

```
library(dplyr)

# Set up a test table
conn <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
artists_df <- data.frame(
         ArtistId = c(1,2),
         Name = c("AC/DC", "The Offspring")
)
DBI::dbWriteTable(conn, "Artist", artists_df)

# Update rows without changing the key.
artists <- tbl(conn, "Artist")
DBI::dbBegin(conn)
y <- data.frame(ArtistId = 1, Name = "DC/AC")
e_rows_update(
     x = artists,
     y = y,
     by = "ArtistId",
     in_place = TRUE)
DBI::dbRollback(conn)

# Update key values of rows.
DBI::dbBegin(conn)
y <- data.frame(ArtistId = 999, Name = "DC/AC")
match <- list(
   x = data.frame("ArtistId" = 1),
   y = data.frame("ArtistId" = 999)
)
e_rows_update(
    x = artists,
    y = y,
    match = match,
```

```
    by = "ArtistId",
    in_place = TRUE)
DBI::dbRollback(conn)
DBI::dbDisconnect(conn)
```

`fillDeductedColumns` *Fill data columns based on foreignTbls*

Description

Fill data columns based on foreignTbls

Usage

```
fillDeductedColumns(tbl, foreignTbls)
```

Arguments

`tbl`	`tbl`
`foreignTbls`	list of foreign tbls as created by `foreignTbl`

Details

When a combination of columns is not found in the foreignTbl, fill the deductedColumns with NA.

on foreignTbls suggesting conflicting data, an arbitrary choice is made. It is best to afterwards check with checkForeignTbls to see if a valid result is obtained.

Value

tbl

Author(s)

Jasper Schelfhout

fixInteger64 *Replace instances of integer64 with actual NA values instead of weird default 9218868437227407266*

Description

Replace instances of integer64 with actual NA values instead of weird default 9218868437227407266

Usage

```
fixInteger64(x)
```

Arguments

x	data.frame

Details

github issue

Value

x with integer64 columns set to bit64::as.integer64(NA)

Author(s)

Jasper Schelfhout

foreignTbl *Create a foreign tibble*

Description

Create a foreign tibble

Usage

```
foreignTbl(
  x,
  y,
  by = intersect(dplyr::tbl_vars(x), dplyr::tbl_vars(y)),
  naturalKey = dplyr::tbl_vars(y),
  allowNew = FALSE
)
```

Arguments

x	tbl. The referencing table.
y	tbl. The referenced table.
by	character. Column names to match on. Note that you should rename and/or typecast the columns in y should they not exactly match the columns in x.
naturalKey	character. The columns that form the natural key in y. These are the only ones that can actually get modified in eDT when changing cells in the table. Reasoning being that these columns should be sufficient to uniquely identify a row in the referenced table. All other columns will be automatically fetched and filled in.
allowNew	logical. Whether or not new values are allowed. If TRUE, the rows in the foreignTbl will only be used as suggestions, not restrictions.

Details

This is a tibble that can be passed onto eDT as a referenced table.

It is the equivalent of a database table to which the data tbl of eDT has a foreign key.

It will be merged with the tbl passed onto the data argument allowing to provide restrictions for certain columns.

Note that row uniqueness for the columns used in by and naturalKey is assumed. This assumption will however not be checked since it is an expensive operation on big datasets. However, if violated, it might give errors or unexpected results during usage of the eDT module.

Value

List with unmodified arguments. However, they have now been checked for validity.

- y, see argument y.
- by, see argument by.
- naturalKey, see argument naturalKey.
- allowNew, see argument allowNew

Author(s)

Jasper Schelfhout

Examples

```
a <- tibble::tibble(
   first_name = c("Albert","Donald","Mickey"),
   last_name_id = c(1,2,2)
 )

b <-  foreignTbl(
 a,
 tibble::tibble(
     last_name = c("Einstein", "Duck", "Mouse"),
```

```
      last_name_id = c(1,2,3)
    ),
 by = "last_name_id",
 naturalKey = "last_name"
)

## Only run this in interactive R sessions
if(interactive()){
  eDT(a,
   foreignTbls = list(b),
   options = list(columnDefs = list(list(visible=FALSE, targets="last_name_id")))
  )
 }
```

`getColumnTypeSums` *Get types of columns in a tbl*

Description

Get types of columns in a tbl

Usage

```
getColumnTypeSums(tbl)
```

Arguments

`tbl`	`tbl`

Value

named list with types of the colums

Author(s)

Jasper Schelfhout

getNonNaturalKeyCols *Get all columns that are not natural keys*

Description

Get all columns that are not natural keys

Usage

```
getNonNaturalKeyCols(foreignTbls)
```

Arguments

foreignTbls	list of foreign tbls as created by foreignTbl

Value

character

Author(s)

Jasper Schelfhout

get_db_table_name *Get name of the tbl in the database*

Description

Get name of the tbl in the database

Usage

```
get_db_table_name(x)
```

Arguments

x	tbl_dbi

Value

SQL, the table name as used in the database

`initData` *Add some extra columns to data to allow for / keep track of modifications*

Description

Add some extra columns to data to allow for / keep track of modifications

Usage

```
initData(
  data,
  ns,
  buttonCol = "buttons",
  statusCol = "status",
  deleteCol = "deleted",
  iCol = "i",
  canDeleteRow = TRUE,
  canEditRow = TRUE
)
```

Arguments

`data`	`data.frame`
`ns`	namespace function
`buttonCol`	`character(1)` name of column with buttons
`statusCol`	`character(1)` name of column with general status (e.g. modified or not).
`deleteCol`	`character(1)` name of the column with deletion status.
`iCol`	`character(1)` name of column containing a unique identifier.
`canDeleteRow`	can be either of the following: • `logical`, e.g. TRUE or FALSE • `function`. Needs as input an argument `row` which accepts a single row `tibble` and as output TRUE/FALSE.
`canEditRow`	can be either of the following: • `logical`, e.g. TRUE or FALSE • `function`. Needs as input an argument `row` which accepts a single row `tibble` and as output TRUE/FALSE.

Value

data with extra columns buttons, status, i.

Author(s)

Jasper Schelfhout

`inputServer` *An input server for a* `data.frame`

Description

An input server for a `data.frame`

Usage

```
inputServer(id, data, ...)
```

Arguments

`id`	`character(1)` module id
`data`	single row `data.frame`
`...`	further arguments for methods

Details

A new method for this can be added if you wish to alter the default behavior of the pop-up modals in `eDT`.

Value

modified version of data

Author(s)

Jasper Schelfhout

Examples

```
if(interactive()){
  library(shiny)
  ui <- inputUI('id')
  server <- function(input,output,session){
    input <- inputServer("id", mtcars[1,])
    observe({print(input())})
  }
shinyApp(ui, server)
}
```

inputServer.default *An input server for a* data.frame

Description

An input server for a data.frame

Usage

```
## Default S3 method:
inputServer(id, data, colnames, notEditable, foreignTbls, ...)
```

Arguments

id	character(1) module id
data	single row data.frame
colnames	named character
notEditable	character columns that should not be edited
foreignTbls	list of foreignTbls. See foreignTbl
...	for compatibility with other methods

Details

Reads all inputs ids that are identical to column names of the data and updates the data.

Value

reactive modified version of data

Author(s)

Jasper Schelfhout

inputUI *An input UI for a* data.frame

Description

An input UI for a data.frame

Usage

```
inputUI(id, ...)
```

Arguments

id	character(1) module id
...	arguments passed onto methods

Details

A new method for this can be added if you wish to alter the default behavior of the pop-up modals in eDT.

Value

HTML. A set of input fields corresponding to the given row.

Author(s)

Jasper Schelfhout

Examples

```
if(interactive()){
  library(shiny)
  ui <- inputUI('id')
  server <- function(input,output,session){
    input <- inputServer("id", mtcars[1,])
    observe({print(input())})
  }
shinyApp(ui, server)
}
```

---

inputUI.default *UI part for modal with input fields for editing*

Description

UI part for modal with input fields for editing

Usage

```
## Default S3 method:
inputUI(id, ...)
```

Arguments

id	character module id
...	for compatibility with method

Details

The UI elements that have an id identical to a column name are used for updating the data.

Value

HTML. A set of input fields corresponding to the given row.

Author(s)

Jasper Schelfhout

`joinForeignTbl` *Merge a tbl with it a foreignTbl*

Description

Merge a tbl with it a foreignTbl

Usage

```
joinForeignTbl(
  tbl,
  foreignTbl,
  keepNA = TRUE,
  by = foreignTbl$by,
  copy = TRUE,
  type = c("inner", "left")[1]
)
```

Arguments

`tbl`	`tbl`
`foreignTbl`	`list` as created by `foreignTbl`
`keepNA`	`logical` keep rows from tbl with NA keys.
`by`	named `character`, columns to join on.
`copy`	`logical`, whether or not to copy the `foreignTbl` to the source of argument `tbl` for joining.
`type`	`character(1)`, type of joint to perform. Can be 'inner' or 'left'.

Details

see also `dplyr` join functions, for example `dplyr::left_join`.

Value

`tbl`, containing both columns from argument `tbl` and argument `foreignTbl`.

Author(s)

Jasper Schelfhout

`overwriteDefaults` *Overwrite default settings with provided settings*

Description

Overwrite default settings with provided settings

Usage

```
overwriteDefaults(defaults, settings)
```

Arguments

`defaults`	named character vector
`settings`	named character vector

Value

named character vector

Author(s)

Jasper Schelfhout

`rollbackTransaction` *Start a transaction for a tibble*

Description

Start a transaction for a tibble

Usage

```
rollbackTransaction(tbl)
```

Arguments

`tbl`	`tbl`

Author(s)

Jasper Schelfhout

rowInsert *Add a row to a table in the database.*

Description

Add a row to a table in the database.

Usage

```
rowInsert(conn, table, values)
```

Arguments

conn	database connection object as given by dbConnect.
table	character
values	named list, row to add. Names are database column names. Unspecified columns will get database defaults.

Value

integer number of affected rows.

rows_delete.dtplyr_step *rows_delete implementation for data.table backends.*

Description

rows_delete implementation for data.table backends.

Usage

```
## S3 method for class 'dtplyr_step'
rows_delete(x, y, by = NULL, ..., unmatched, copy = FALSE, in_place = FALSE)
```

Arguments

x, y	A pair of data frames or data frame extensions (e.g. a tibble). y must have the same columns of x or a subset.
by	An unnamed character vector giving the key columns. The key columns must exist in both x and y. Keys typically uniquely identify each row, but this is only enforced for the key values of y when rows_update(), rows_patch(), or rows_upsert() are used. By default, we use the first column in y, since the first column is a reasonable place to put an identifier variable.

...	Other parameters passed onto methods.
unmatched	For rows_update(), rows_patch(), and rows_delete(), how should keys in y that are unmatched by the keys in x be handled? One of: • "error", the default, will error if there are any keys in y that are unmatched by the keys in x. • "ignore" will ignore rows in y with keys that are unmatched by the keys in x.
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
in_place	Should x be modified in place? This argument is only relevant for mutable backends (e.g. databases, data.tables). When TRUE, a modified version of x is returned invisibly; when FALSE, a new object representing the resulting changes is returned.

Value

An object of the same type as x. The order of the rows and columns of x is preserved as much as possible. The output has the following properties:

- rows_update() and rows_patch() preserve the number of rows; rows_insert(), rows_append(), and rows_upsert() return all existing rows and potentially new rows; rows_delete() returns a subset of the rows.
- Columns are not added, removed, or relocated, though the data may be updated.
- Groups are taken from x.
- Data frame attributes are taken from x.

If in_place = TRUE, the result will be returned invisibly.

Author(s)

Jasper Schelfhout

---

rowUpdate *Update rows in the database.*

Description

Update rows in the database.

Usage

```
rowUpdate(conn, table, values, where)
```

Arguments

conn	database connection object as given by dbConnect.
table	character
values	named list, values to be set. Names are database column names.
where	named list, values to filter on. Names are database column names. If NULL no filter is applied.

Value

integer number of affected rows.

---

runDemoApp *Run a demo app*

Description

Run a demo app

Usage

```
runDemoApp(app = "database", ...)
```

Arguments

app	demoApp to run. Options: database / mtcars / custom
...	arguments passed onto the demoApp

Details

These apps are for illustrative purposes.

Value

An object that represents the app. Printing the object or passing it to runApp() will run the app.

Examples

```
## Only run this example in interactive R sessions
if(interactive()){

  # Database
  tmpFile <- tempfile(fileext = ".sqlite")
  file.copy(system.file("extdata", "chinook.sqlite", package = 'editbl'), tmpFile)

  conn <- connectDB(dbname = tmpFile)

  runDemoApp(app = "database", conn = conn)
```

```
  DBI::dbDisconnect(conn)

  unlink(tmpFile)

  # mtcars
  runDemoApp(app = "mtcars")

  # Any tibble of your liking
  runDemoApp(app = "custom", dplyr::tibble(iris))
}
```

`runDemoApp_custom` *Run a custom demo app*

Description

Run a custom demo app

Usage

```
runDemoApp_custom(x)
```

Arguments

x	tbl

Value

An object that represents the app. Printing the object or passing it to `runApp()` will run the app.

`runDemoApp_DB` *Run a demo app*

Description

Run a demo app

Usage

```
runDemoApp_DB()
```

Value

An object that represents the app. Printing the object or passing it to `runApp()` will run the app.

runDemoApp_mtcars *Run a demo app*

Description

Run a demo app

Usage

```
runDemoApp_mtcars()
```

Value

An object that represents the app. Printing the object or passing it to `runApp()` will run the app.

runDevApp *Run a development app*

Description

Run a development app

Usage

```
runDevApp()
```

Details

This app prints some of the server objects and has a button to interactively browse the code. This is useful for debugging and experimenting with new features.

Value

An object that represents the app. Printing the object or passing it to `runApp()` will run the app.

`selectInputDT_Server` *Server part to use a* `datatable` *as select input*

Description

Server part to use a `datatable` as select input

Usage

```
selectInputDT_Server(
  id,
  label = "",
  choices,
  selected = NULL,
  multiple = FALSE
)
```

Arguments

`id`	`character(1)` same one as used in `selectInputDT_UI`
`label`	`character(1)`
`choices`	`data.frame`
`selected`	`data.frame` with rows available in `choices`.
`multiple`	`logical`. Whether or not multiple row selection is allowed

Value

A selection of rows from the `data.frame` provided under choices.

Author(s)

Jasper Schelfhout

See Also

`shiny::selectInput`. This function can be more convenient for selecting rows with multiple columns.

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  ui <- selectInputDT_UI('id')
  data <- data.frame(id = 1:3, name = letters[1:3])
  server <- function(input,output, session){
    selected = selectInputDT_Server('id', choices = data, selected = data[1,] )
    observe({print(selected())})
```

```
  }
  shiny::shinyApp(ui, server)

}
```

`selectInputDT_UI` *UI part of a DT select input*

Description

UI part of a DT select input

Usage

```
selectInputDT_UI(id)
```

Arguments

`id`	`character(1)` same one as used in `selectInputDT_Server`

Value

HTML

Author(s)

Jasper Schelfhout

Examples

```
## Only run this example in interactive R sessions
if(interactive()){
  ui <- selectInputDT_UI('id')
  data <- data.frame(id = 1:3, name = letters[1:3])
  server <- function(input,output, session){
    selected = selectInputDT_Server('id', choices = data, selected = data[1,] )
    observe({print(selected())})
  }
  shiny::shinyApp(ui, server)

}
```

shinyInput *Get a shiny input for a column of a* tbl

Description

Get a shiny input for a column of a tbl

Usage

```
shinyInput(x, inputId, label, selected)
```

Arguments

x	column
inputId	shiny input Id
label	character(1)
selected	object of class of x

Value

shiny input

Author(s)

Jasper Schelfhout

standardizeArgument_colnames *Standardize colnames argument to the format of named character vector*

Description

Standardize colnames argument to the format of named character vector

Usage

```
standardizeArgument_colnames(colnames, data)
```

Arguments

colnames	if missing, the column names of the data; otherwise it can be an unnamed character vector of names you want to show in the table header instead of the default data column names; alternatively, you can provide a *named* numeric or character vector of the form `'newName1' = i1,'newName2' = i2` or `c('newName1' = 'oldName1', 'newName2' = 'oldName2', ...)`, where `newName` is the new name you want to show in the table, and `i` or `oldName` is the index of the current column name
data	`tbl`. The function will automatically cast to tbl if needed.

Value

named character vector

Author(s)

Jasper Schelfhout

---

standardizeArgument_editable

Standardized editable argument to be in the form of a list

Description

Standardized editable argument to be in the form of a list

Usage

```
standardizeArgument_editable(editable, data)
```

Arguments

editable	`FALSE` to disable the table editor, or `TRUE` (or `"cell"`) to enable editing a single cell. Alternatively, you can set it to `"row"` to be able to edit a row, or `"column"` to edit a column, or `"all"` to edit all cells on the current page of the table. In all modes, start editing by doubleclicking on a cell. This argument can also be a list of the form `list(target = TARGET, disable = list(columns = INDICES))`, where `TARGET` can be `"cell"`, `"row"`, `"column"`, or `"all"`, and `INDICES` is an integer vector of column indices. Use the list form if you want to disable editing certain columns. You can also restrict the editing to accept only numbers by setting this argument to a list of the form `list(target = TARGET, numeric = INDICES)` where `INDICES` can be the vector of the indices of the columns for which you want to restrict the editing to numbers or `"all"` to restrict the editing to numbers for all columns. If you don't set `numeric`, then the editing is restricted to numbers for all numeric columns; set `numeric = "none"` to disable this behavior. It is also possible to edit the cells in text areas, which are useful for large contents. For that, set the `editable` argument to a

	list of the form `list(target = TARGET, area = INDICES)` where `INDICES` can be the vector of the indices of the columns for which you want the text areas, or `"all"` if you want the text areas for all columns. Of course, you can request the numeric editing for some columns and the text areas for some other columns by setting `editable` to a list of the form `list(target = TARGET, numeric = INDICES1, area = INDICES2)`. Finally, you can edit date cells with a calendar with `list(target = TARGET, date = INDICES)`; the target columns must have the `Date` type. If you don't set `date` in the `editable` list, the editing with the calendar is automatically set for all `Date` columns.
`data`	`tbl`. The function will automatically cast to tbl if needed.

Value

list of the form `list(target = foo, ...)`

Author(s)

Jasper Schelfhout

`whereSQL` *Generate where sql*

Description

Generate where sql

Usage

```
whereSQL(conn, table, column, operator = "in", values = NULL)
```

Arguments

`conn`	database connection object as given by `dbConnect`.
`table`	character table name (or alias used in query)
`column`	character column of table
`operator`	character
`values`	character vector of values

Value

character sql

Author(s)

Jasper Schelfhout

Index