Package 'DTedit'

June 20, 2020

Title Editable DataTables for Shiny Apps

Version 2.2.0

Description Extends DT::DataTable to allow users to create, edit, and delete rows from the data table.

Depends R (>= 3.3), DT, shiny, blob, methods

Suggests RSQLite,
base64enc,
randomNames,
knitr,
rmarkdown,
testthat,
shinytest,
covr,
devtools,
withr

License LGPL

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0.9000

VignetteBuilder knitr

Collate 'DTedit-package.R'
'dtedit.R'
'dtedit_demo.R'
'dtedit_test.R'

R topics documented:

DTedit-package *Editable DataTables for shiny apps*

Description

Editable DataTables for shiny apps

Author(s)

<jason@bryer.org>

dtedit *Create a DataTable with Add, Edit and Delete buttons.*

Description

dtedit - editable DataTable

dteditmod - editable DataTable, adapted for use in modules

Usage

```
dtedit(input, output, name, thedata, ...)

dteditmod(
  input,
  output,
  session,
  thedata,
 view.cols = names(shiny::isolate(if (shiny::is.reactive(thedata)) {    thedata() }
   else {     thedata })),
 edit.cols = names(shiny::isolate(if (shiny::is.reactive(thedata)) {    thedata() }
   else {     thedata })),
  edit.label.cols = edit.cols,
  delete.info.cols = view.cols,
  delete.info.label.cols = delete.info.cols,
  input.types,
  input.choices = NULL,
  input.choices.reactive = NULL,
  action.buttons = NULL,
  selectize = TRUE,
  modal.size = "m",
  text.width = "100%",
  textarea.width = "570px",
  textarea.height = "200px",
  date.width = "100px",
  numeric.width = "100px",
  select.width = "100%",
  defaultPageLength = 10,
  max.fileInputLength = 1e+08,
```

```
  title.delete = "Delete",
  title.edit = "Edit",
  title.add = "New",
  label.delete = "Delete",
  label.edit = "Edit",
  label.add = "New",
  label.copy = "Copy",
  label.save = "Save",
  label.cancel = "Cancel",
  text.delete.modal = "Are you sure you want to delete this record?",
  show.delete = TRUE,
  show.update = TRUE,
  show.insert = TRUE,
  show.copy = TRUE,
  callback.delete = function(data, row) { },
  callback.update = function(data, olddata, row) { },
  callback.insert = function(data, row) { },
  callback.actionButton = function(data, row, buttonID) { },
  click.time.threshold = 2,
  datatable.options = list(pageLength = defaultPageLength),
  datatable.rownames = FALSE,
  datatable.call = function(...) {     DT::datatable(...) },
  ...
)
```

Arguments

input	Shiny input object passed from the server.
output	Shiny output object passed from the server.
name	(name is available in dtedit only). The name of the outputted editable datatable. The name passed to dtedit is the same as the name passed to uiOutput. Put uiOutput(name) where you want the editable datatable in the ui.R. When using more than one dtedit within a Shiny application the name must be unique. (name is converted to the session argument of dteditmod.)
thedata	a data frame to view and edit. can be a reactive
...	arguments not recognized by DTedit are passed to DT::renderDataTable By default, datatable.call uses DT::dataframe, so this limits the options that can be passed through this method.
session	Shiny session object (an environment) passed from the server. Alternatively, the 'name' (character) of the outputted editable datatable.
view.cols	character vector with the column names to show in the DataTable. This can be a subset of the full data.frame.
edit.cols	character vector with the column names the user can edit/add. This can be a subset of the full data.frame.
edit.label.cols	character vector with the labels to use on the edit and add dialogs. The length and order of code.cols.labels must correspond to edit.cols.
delete.info.cols	character vector with the column names specifying which values are presented on the delete dialog. This can be a subset of the full data.frame. Defaults to view.cols. If NULL, no data values are shown on the delete dialog.

`delete.info.label.cols`
: character vector with the labels to use on the delete dialog. The length and order of `delete.info.label.cols` must correspond to `delete.info.cols`.

`input.types`
: a character vector where the name corresponds to a column in `edit.cols` and the value is the input type. Possible values are:

  - `dateInput` - input changed by `date.width`
  - `selectInput` - choices determined by `input.choices`, or the levels of the data column variable (if the column variable is of class `factor`), or the already present values in the data column.
  - `selectInputMultiple` - choices determined by `input.choices` or the already present values in the data column.
  - `selectInputReactive` - choices determined by a reactive variable, as defined by `input.choices` and `input.choices.reactive`.
  - `selectInputMultipleReactive` - choices determined by a reactive variable, as defined by `input.choices` and `input.choices.reactive`
  - `numericInput` - input changed by `numeric.width`
  - `textInput` - input changed by `text.width`
  - `textAreaInput` - input changed by `textarea.width` and `textarea.height`
  - `passwordInput`
  - `fileInput` - type of acceptable file types is defined by `input.choices`. Maximum file length is modifed by `max.fileInputLength`

  One case where this parameter is desirable is when a text area is required instead of a simple text input.

`input.choices`
: a list of character vectors. The names of each element in the list must correspond to a column name in the data. The value, a character vector, are the options presented to the user for data entry, in the case of input type `selectInput`).

  In the case of input type `selectInputReactive` or 'selectInputMultipleReactive", the value is the name of the reactive in 'input.choices.reactive'

  In the case of input type 'fileInput" this is the 'accept' argument, which specifies the type of file which is acceptable. Can be a case insensitive file extension (e.g. '.csv' or '.rds') or a MIME type (e.g. 'text/plain' or 'application/pdf').

`input.choices.reactive`
: a named list of reactives, referenced in 'input.choices' to use for input type `selectInputReactive` or `selectInputMultipleReactive`. The reactive itself is a character vector.

`action.buttons`
: a named list of action button columns. Each column description is a list of `columnLabel`, `buttonLabel`, `buttonPrefix`, `afterColumn`.

  - `columnLabel` label used for the column.
  - `buttonLabel` label used for each button
  - `buttonPrefix` used as the prefix for action button IDs. The suffix will be a number from '1' to the number of rows. Prefix and suffix will be separated with an underscore '_'.
  - `afterColumn` if provided, the action button column is placed after this named column.

`selectize`
: Whether to use selectize.js or not. See `selectInput` for more info.

`modal.size`
: the size of the modal dialog. See `modalDialog`.

`text.width`
: width of text inputs.

`textarea.width` the width of text area inputs.

`textarea.height`
: the height of text area inputs.

`date.width` the width of data inputs

`numeric.width` the width of numeric inputs.

`select.width` the width of drop down inputs.

`defaultPageLength`
: number of rows to show in the data table by default.

`max.fileInputLength`
: the maximum length (in bytes) of `fileInput`. Shiny itself has a default limit of 5 megabytes per file. The limit can be modified by using shiny.maxRequestSize option.

`title.delete` the title of the dialog box for deleting a row.

`title.edit` the title of the dialog box for editing a row.

`title.add` the title of the dialog box for inserting a new row.

`label.delete` the label of the delete button.

`label.edit` the label of the edit button.

`label.add` the label of the add button.

`label.copy` the label of the copy button.

`label.save` the label of the save button.

`label.cancel` the label of the cancel button.

`text.delete.modal`
: the text shown in the delete modal dialog.

`show.delete` whether to show/enable the delete button.

`show.update` whether to show/enable the update button.

`show.insert` whether to show/enable the insert button.

`show.copy` whether to show/enable the copy button.

`callback.delete`
: a function called when the user deletes a row. This function should return an updated data.frame.

`callback.update`
: a function called when the user updates a row. This function should return an updated data.frame.

`callback.insert`
: a function called when the user inserts a new row. This function should return an updated data.frame.

`callback.actionButton`
: a function called when the user clicks an action button. called with arguments `data`, `row` and `buttonID`. This function can return an updated data.frame, alternatively return NULL if data is not to be changed.

`click.time.threshold`
: This is to prevent duplicate entries usually by double clicking the save or update buttons. If the user clicks the save button again within this amount of time (in seconds), the subsequent click will be ignored. Set to zero to disable this feature. For developers, a message is printed using the warning function.

`datatable.options`
: options passed to `DT::renderDataTable`. See https://rstudio.github.io/DT/options.html for more information.

`datatable.rownames`
: show rownames as part of the datatable? `TRUE` or `FALSE`. Note that if datatable.call includes DT::format* calls, then `datatable.rownames` must equal `TRUE`

`datatable.call`
: pre-processing call when calling `DT::renderDataTable`. Can be defined, for example, to include DT::format* calls. `dtedit` will pass several arguments to the `datatable.call` function.

  - `data` a dataframe. may have been processed to add `actionButtons`
  - `options` - `datatable.options`
  - `rownames` - `datatable.rownames`
  - `escape` - escape all columns except those with action buttons.
  - `selection` - `single`

Details

`dtedit` is used in conjunction with `uiOutput` to create editable datatables. `dtedit` is used in a shiny application's server definition, `uiOutput` is used in the UI (user interface) definition.

`dteditmod` is used in conjunction with `callModule` and `dteditmodUI` to create editable datatables in a module environment. `dteditmod` is called through `callModule` in the 'server' section of the shiny application. `dteditmodUI` is called in the 'UI' (user-interface) section of the shiny app.

This object will maintain data state. However, in order of the data to persist between Shiny instances, data needs to be saved to some external format (e.g. database or R data file). The callback functions provide a mechanism for this function to interact with a permanent data storage scheme. The callback functions are called when the user adds, updates, or deletes a row from the data table. The callback must accept two parameters: `data` and `row`. For inserting and updating, the `data` object is the current state of data table including any additions or updates. The `row` parameter indicates which row from `data` was modified (or added). For deletions, however, the `data` represents the data table just before deleting the specified row. That is, if `callback.delete` returns a `data.frame`, that will be the new data table; otherwise this function will remove row `row` from `data` and that will become the current data table.

The callback functions may throw errors (see e.g. `stop`) if there are problems with data. That is, if data validation checks indicate data problems before inserting or updating a row the function may throw an error. Note that the error message will be presented to the user so providing messages meaningful to the user is recommended. Moreover, if an error is thrown, the modal dialog is not dismissed and the user can further edit the data and retry the insertion or update.

Callback functions may return a `data.frame`. When a `data.frame` is returned that will become the current state of the data table. If anything else is returned then the internal `data.frame` will be used.

Value

Returns reactiveValues

- $theData - the current state of `DTedit`'s copy of the data
- $view.cols
- $edit.cols
- $edit.count - number of edits to data done within `DTedit` (does not include changes to `DTedit`'s copy of the data secondary to changes of a reactive `thedata`)

See Also

- `example("dtedit")` a simple example.
- `dtedit_demo()` demonstration of dtedit.
- `dtedit_reactive_demo()` reactive dataframe
- `dtedit_selectInputReactive_demo()` reactive selectInput
- `dteditmodUI` : the companion user-interface function for `dteditmod`.
- `example("dteditmodUI")` a simple module example with reactive dataframe
- `dteditmod_demo()` a more complex module example. Database interaction and interactions between the data of multiple datatables.
- `dteditmod_fileInput_demo()` a modular example including binary file input and action buttons.

Other Datatable Edit functions: `dteditmodUI()`

Examples

```
# minimal DTedit example 'dtedit'
# you can try this example in interactive mode
# with 'example("dtedit")'

library(shiny)
library(DTedit)

server <- function(input, output) {

  Grocery_List <- dtedit(
    input, output,
    name = 'Grocery_List',
    thedata = data.frame(
      Buy = c('Tea', 'Biscuits', 'Apples'),
      Quantity = c(7, 2, 5),
      stringsAsFactors = FALSE
    )
  )
}

ui <- fluidPage(
  h3('Grocery List'),
  uiOutput('Grocery_List')
)

if (interactive())
  shinyApp(ui = ui, server = server)

#### end of 'dtedit' example ####


# minimal DTedit example 'dteditmod'
# this is a separate application from the 'dtedit' example!
#
# unfortunately, this application cannot be
# tried with 'example("dteditmod")', but you can copy
```

```
# and paste to execute in 'interactive' console mode,
# or copy the lines into an '.R' file and choose
# 'Run App' from RStudio.
library(shiny)
library(DTedit)

server <- function(input, output, session) {

  Grocery_List <- callModule(
    dteditmod,
    id = 'Grocery_List',
    thedata = data.frame(
      Buy = c('Tea', 'Biscuits', 'Apples'),
      Quantity = c(7, 2, 5),
      stringsAsFactors = FALSE
    )
  )
}

ui <- fluidPage(
  h3('Grocery List'),
  dteditmodUI('Grocery_List')
)

if (interactive() || isTRUE(getOption("shiny.testmode")))
  shinyApp(ui = ui, server = server)
```

`dteditmodUI` *Create a DataTable with Add, Edit and Delete buttons.*

Description

dteditmodUI - user-interface function for module use

Usage

```
dteditmodUI(id)
```

Arguments

`id`	the namespace of the module

Details

Use in conjunction with `callModule` and `dtedit` to create editable datatables. `dteditUI` is used in the 'user interface' component of the shiny app.

See Also

`dteditmod` : the companion server-component function.

- `example("dteditmodUI")` a simple example with a reactive dataframe

- `dteditmod_demo()` a more complex example. Includes database interaction and interactions between the data of multiple datatables.

Other Datatable Edit functions: `dtedit()`

Examples

```
##### Minimal DTedit example using reactive dataframe #####
library(shiny)
library(DTedit)

##### Create the Shiny server #####
server <- function(input, output) {

  data <- reactiveVal() # # 'data' will be a 'reactive' dataframe
  data(data.frame(Column1 = c("Apple", "Cherry", "Frozen"),
                  Column2 = c("Pie", "Tart", "Yoghurt"),
                  stringsAsFactors = FALSE))
  data_DT_gui <- callModule(
    dteditmod,
    'dataspace',
    thedata = data,
    edit.cols = c("Column1", "Column2")
  )

  observe({
    data(
      isolate(
        as.data.frame(
          data_DT_gui$thedata,
          stringsasfactors = FALSE
        )
      )
    )
    print(isolate(data()))
    print(paste("Edit count:", data_DT_gui$edit.count))
    # only reacts to change in $edit.count
  })

  observeEvent(input$data_scramble, {
    print("Scrambling...")
    temp <- data()
    if (nrow(temp)>0) {
      row <- sample(1:nrow(temp), 1)  # row
      col <- sample(1:2, 1)           # column
      temp[row, col] <- paste(
        sample(unlist(strsplit(temp[row, col], "")),
               nchar(temp[row, col])),
        sep = '', collapse = '')
      data(temp) # adjusted dataframe 'automatically' read by DTedit
    }
  })
}

##### Create the shiny UI ######
ui <- fluidPage(
  h3("DTedit using reactive dataframe"),
```

```
  wellPanel(p("Try the 'Scramble' button!")),
  dteditmodUI("dataspace"),
  actionButton("data_scramble", "Scramble an entry")
)

if (interactive() || isTRUE(getOption("shiny.testmode")))
  shinyApp(ui = ui, server = server)
```

`dteditmod_demo` *Run a shiny app showing how the DTedit function works.*

Description

modularized version of dtedit_demo and dtedit_selectInputReactive_demo. Uses dteditmod/dteditmodUI version of dtedit.

Usage

```
dteditmod_demo(...)
```

Arguments

`...`	arguments pass to `runApp` For example, can launch in showcase mode: `DTedit::dteditmod_demo(display.mode = "showcase")` Note that 'showcase' mode shows all the .R files in the 'shiny_demo' directory, not just the .R file used for this demonstration!

Details

Demonstrates adding/editing/deleting data rows, callbacks, interaction with database.

Demonstrates interaction between datatables using reactives

`dteditmod_fileInput_demo` *Run a shiny app showing how the DTedit function works.*

Description

Demonstrates file input, blobs and action buttons

Usage

```
dteditmod_fileInput_demo(...)
```

Arguments

...	arguments pass to runApp For example, can launch in showcase mode: DTedit::dteditmod_fileInput_demo(display.mode = "showcase") Note that 'showcase' mode shows all the .R files in the 'shiny_demo' directory, not just the .R file used for this demonstration!

Details

(modularized, uses dteditmod/dteditmodUI)

---

dtedit_demo *Run a shiny app showing how the DTedit function works.*

Description

Demonstrates adding/editing/deleting data rows, callbacks, interacting with a database, selectInput and selectInputMultiple.

Usage

```
dtedit_demo(...)
```

Arguments

...	arguments pass to runApp For example, can launch in showcase mode: DTedit::dtedit_demo(display.mode = "showcase") Note that 'showcase' mode shows all the .R files in the 'shiny_demo' directory, not just the .R file used for this demonstration!

---

dtedit_reactive_demo *Run a shiny app showing how the DTedit function works.*

Description

reactive dataframe

Usage

```
dtedit_reactive_demo(...)
```

Arguments

...	arguments pass to runApp For example, can launch in showcase mode: DTedit::dtedit_reactive_demo(display.mode = "showcase") Note that 'showcase' mode shows all the .R files in the 'shiny_demo' directory, not just the .R file used for this demonstration!

dtedit_selectInputReactive_demo

Run a shiny app showing how the DTedit function works.

Description

demonstrates interaction between datatables using selectInputReactive and selectInputMultipleReactive

Usage

```
dtedit_selectInputReactive_demo(...)
```

Arguments

...	arguments pass to runApp For example, can launch in showcase mode: `DTedit::dtedit_selectInputReactive_demo(display.mode = "showcase")` Note that 'showcase' mode shows all the .R files in the 'shiny_demo' directory, not just the .R file used for this demonstration!

dtedit_test

test application

Description

for testthat/codecov

Usage

```
dtedit_test(appname = "simple", ...)
```

Arguments

appname	choose test simple simple_modular reactive callback error_test selectInputReactive password
...	extra options passed to shiny::shinyApp

Value

a shiny app

Index