Package 'shinyQueryBuilder'

July 23, 2025

Type Package

Title Construct Complex Filtering Queries in 'Shiny'

Version 0.1.0

Maintainer Krystian Igras <krystian8207@gmail.com>

Description Input widget that allows to construct complex filtering queries in 'Shiny'. It's a wrapper for 'JavaScript' library 'jQuery-QueryBuilder', check <https://querybuilder.js.org/>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

Depends queryBuilder

Imports rlang, R6, magrittr, jsonlite, htmltools, shiny, glue, purrr

Collate 'shinyQueryBuilder-package.R' 'query_builder_input.R' 'filters.R' 'operators.R' 'query_from_data.R'

Suggests knitr, rmarkdown

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Krystian Igras [aut, cre], Damien Sorel [cph] (jQuery-QueryBuilder)

Repository CRAN

Date/Publication 2024-09-26 14:00:02 UTC

Contents

`shinyQueryBuilder-package` *Shiny Wrapper for jQuery-QueryBuilder*

Description

Shiny Wrapper for jQuery-QueryBuilder

`genQueryFilters` *Generate filters definition*

Description

Generate filters definition

Usage

```
genQueryFilters(
  data,
  settings = list(),
  .queryBuilderConfig = queryBuilder::queryBuilderConfig
)
```

Arguments

`data`	Dataset from which filters should extracted.
`settings`	Named list. Column-specific filter configuration. For each variable the provided settings will overwrite the default ones.
`.queryBuilderConfig`	R6 object of class 'queryBuilderConfig' storing queryOperators. See query-operator.

Value

Nested list object storing generated filters configuration.

Examples

```
genQueryFilters(
  iris,
  list(
    Species = list(operators = c("equal", "not_equal"))
  )
)
```

js *Store JS definition as character string*

Description

Store JS definition as character string

Usage

```
js(x)
```

Arguments

x	Character string containing valid JS object e.g. function

Value

An object of class 'json' storing the provided character string.

query-operators *Configure available user interface operators*

Description

Configure available user interface operators

Usage

```
mapOperator(
  name,
  apply_to,
  optgroup = "basic",
  nb_inputs = 1,
  multiple = FALSE,
  .queryBuilderConfig = queryBuilder::queryBuilderConfig
)

listMappedOperators(
  r_class,
  print = TRUE,
  .queryBuilderConfig = queryBuilder::queryBuilderConfig
)
```

Arguments

`name`	Name of the operator to be mapped.
`apply_to`	Precise what field types (classes) should the operator be available to. When `operators` is not defined for queryFilter, all of the operators matching 'queryFilter' type will be available in the operators dropdown. Possible values are 'character', 'factor', 'integer', 'numeric', 'POSIXct', 'Date' and 'logical'.
`optgroup`	Character string ("basic" default). Operators with the same 'optgroup' will be presented within a separate group in the operators dropdown.
`nb_inputs`	Integer. The number of inputs displayed. See 'Details' for more information.
`multiple`	Logical. Inform the builder if operator can accept multiple values for associated inputs. In order to enable multiple values for specific input, set 'multiple = TRUE' when creating queryFilters.
`.queryBuilderConfig`	R6 object of class 'queryBuilderConfig' storing queryOperators. See queryoperator.
`r_class`	Optional R class to list operators assigned to it. When skipped all the mapped operators will be summed up.
`print`	Should the operators summary be printed?

Details

When configuring a single query rule, user needs to precise three values in 'queryBuilderInput' interface:

- 1. field - Name of the field that can be interpreted as a filtered column name. Selected with dropdown.
- 2. operator - Name of the operator to be applied to the field. Selected with dropdown.
- 3. operator value(s) - Value(s) that narrows down the operator definition. Depending on the chosen operator, such input can be take through various kind of **input controllers**.

More detailed configuration for operators linked to specific fields as long as **input controllers** for taking operator values should be set with queryFilter.

`mapOperator` is responsible to establish connection between user interface operators and queryOperator, that are responsible to convert user input to a valid R-expression. The provided configuration allows to shape what **input controllers** should be used to allow users providing operators' value(s).

Parameter 'multiple' precises whether queryBuilderInput should allow to provide multiple values for each input controller. When input controller accepts more than one value and user provides them, in case of 'multiple = FALSE', 'queryBuilderInput' will alert about it and won't send any values to application server.

Please remember 'multiple = TRUE', doesn't mean the associated input controller will automatically accept multiple values, this needs to be separately set for each queryFilter, that is responsible for input controllers configuration.

Parameter 'nb_inputs' informs how many input controllers should be rendered to take operator value(s).

A good practice is to configure your operators the following way:

- `nb_inputs = 0` - Operator associated function doesn't require any value, e.g. 'is_null' or 'is_empty' that only require 'field' name.
- `nb_inputs = n, multiple = FALSE` - Operator associated function requires exactly 'n' values, e.g. 'n=2' for 'between' that requires lower and upper bound to precise it. As a result 'n' separate input controllers will be rendered, each taking a single value.
- `nb_inputs = 1, multiple = TRUE` - Operator associated function accepts dynamic number of values, e.g. 'in'. As a result one single input controller will be rendered, and operator will allow it to have multiple values set.

Value

No return value, called for side effects.

List of operators registered within `.queryBuilderConfig`.

Examples

```
# Set backend operator
in_closed_range <- function(field, bounds) {
  field >= bounds[1] & field <= bounds[2]
}
queryBuilder::setQueryOperators(
  within = queryBuilder::queryOperator(in_closed_range)
)

queryBuilder::listQueryOperators()

# Map backend operator to the user interface one

mapOperator(
  name = "within",
  nb_inputs = 2, # take value with 2 input controllers
  multiple = FALSE, # verify if only single value per controller is set
  apply_to = c("numeric", "Date", "logical") # apply operator to selected field types
)

listMappedOperators()

filters = list(
  queryFilter(
    "Sepal.Length", operators = c("within", "less"),
    type = "numeric", values = range(iris$Sepal.Length)
  ),
  # no operators set, means take all for "character"
  queryFilter("Species", type = "character", values = levels(iris$Species))
)

ui <- shiny::fluidPage(
  title = title,
  queryBuilderInput(
    "qb",
    filters = filters
```

```
  ),
  shiny::verbatimTextOutput("expr")
)

server <- function(input, output, session) {
  output$expr <- shiny::renderPrint({
    print(queryToExpr(input$qb))
  })
}

if (interactive()) {
  shiny::shinyApp(ui, server)
}
```

---

queryBuilderInput *Generate Shiny Query Widget*

Description

Generate Shiny Query Widget

Usage

```
queryBuilderInput(
  inputId,
  filters,
  rules = list(),
  operators = NULL,
  optgroups,
  default_filter,
  sort_filters = FALSE,
  allow_groups = TRUE,
  allow_rm_groups = TRUE,
  allow_empty = TRUE,
  display_errors = TRUE,
  conditions = c("AND", "OR"),
  default_condition = "AND",
  inputs_separator = " , ",
  display_empty_filter = TRUE,
  select_placeholder = "------",
  lang,
  plugins,
  allow_add_rules = TRUE,
  allow_rm_rules = TRUE,
  .queryBuilderConfig = queryBuilder::queryBuilderConfig
)
```

```
updateQueryBuilderInput(
  session,
  inputId,
  rules,
  filters,
  allow_add_rules,
  allow_rm_rules,
  allow_groups,
  allow_rm_groups
)
```

Arguments

inputId	The input slot that will be used to access the value.
filters	(required) List of available filters in the builder. See queryFilter.
rules	Initial set of rules set with 'queryBuilder' package. See queryGroup and queryRule. For 'queryRule', 'shinyQueryBuilder' accepts an extra argument 'flags', that consists of four logical elements: 'filter_readonly', 'operator_readonly', 'value_readonly' and 'no_delete'. These options prevent from changing the rule inputs and removing the rule in the controller. For 'queryGroup', 'shinyQueryBuilder' accepts an extra argument 'flags', that consists of four logical elements: 'condition_readonly', 'no_add_rule', 'no_add_group' and 'no_delete'. These options allow to disable corresponding group management options.
operators	Vector of operator names that should be limited in the input. Leave NULL to allow all of the configured filters.
optgroups	Named list. Configuration of labels for filters and operators. List names should consists of 'optgroup' ids, whereas values, the desired labels to be displayed.
default_filter	Character string. The id of the default filter chosen for any new rule.
sort_filters	Set to 'TRUE' to sort filters alphabetically.
allow_groups	Logical or integer. Number of allowed nested groups. TRUE for no limit.
allow_rm_groups	Logical. Should removing groups be enabled.
allow_empty	Logical. No error will be thrown if the builder is entirely empty.
display_errors	Logical ('TRUE', default). When an error occurs on a rule, display an icon with a tooltip explaining the error.
conditions	Character vector. Array of available group conditions. In order to create custom condition check setQueryConditions.
default_condition	Character string. Default active condition selected for each new group.
inputs_separator	Character string that will be used to separate multiple input controllers for operator values (for operators with 'nb_inputs > 1'). Default is ','.
display_empty_filter	Logical. Add an empty option with 'select_placeholder' string to the filter dropdowns. If the empty filter is disabled and no default_filter is defined, the first filter will be loaded when adding a rule.

`select_placeholder`
: Character string. An option that can be chosen to select empty filter.

`lang`
: Nested named list providing language translations for selected controller labels. See https://github.com/mistic100/jQuery-QueryBuilder/blob/dev/src/i18n/en.json for the required structure, or load one of the existing files included at https://github.com/mistic100/jQuery-QueryBuilder/tree/dev/src/i18n.

`plugins`
: List of plugins names used for the widget. See https://querybuilder.js.org/plugins.html.

`allow_add_rules`
: Logical. Should adding new rules be enabled.

`allow_rm_rules`
: Logical. Should removing rules be enabled.

`.queryBuilderConfig`
: R6 object of class 'queryBuilderConfig' storing queryOperators. See queryOperator.

`session`
: Shiny session object.

Value

Nested list of 'shiny.tag' objects, defining html structure of the input, or no value in case of usage of 'updateQueryBuilderInput' method.

Examples

```
ui <- shiny::fluidPage(
  queryBuilderInput(
    "qb",
    filters = list(
      queryFilter(
        "Species", type = "character", operators = c("in", "equal"),
        values = levels(iris$Species), multiple = TRUE,
        optgroup = "char_fields"
      ),
      queryFilter(
        "Sepal.Length", type = "numeric",
        values = range(iris$Sepal.Length), optgroup = "num_fields"
      )
    ),
    rules = queryGroup(
      condition = "AND",
      queryRule("Species", "equal", "setosa", flags = list(no_delete = TRUE)),
      queryRule("Sepal.Length", "between", c(5, 7))
    ),
    optgroups = list(num_fields = "Numerical fields", char_fields = "Character fields")
  ),
  shiny::verbatimTextOutput("expr")
)
server <- function(input, output, session) {}

if (interactive()) {
```

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

---

queryFilter *Define query filter.*

Description

Filters are responsible for defining available options for providing field-rules in the interface. With filters you may decide what operators should be available for the field, what possible operator-values can be chosen or even customize what kind of input controllers should be used for that goal.

Usage

```
queryFilter(
  id,
  field,
  label,
  optgroup,
  type,
  input,
  values,
  value_separator,
  default_value,
  input_event,
  size,
  rows,
  multiple,
  placeholder,
  vertical,
  validation,
  operators,
  default_operator,
  plugin,
  plugin_config,
  data,
  valueSetter,
  valueGetter,
  unique
)
```

Arguments

id	Character string (required). Unique identifier of the filter.
field	Character string (equals ‘id‘ when missing). Field used by the filter, multiple filters can use the same field. The provided field wiil be used in the returned query.

`label` Character string (equals 'field' when missing). Label used to display the field.

`optgroup` Fields with the same 'optgroup' will be presented within a separate group in the fields dropdown. If skipped, the field will be not listed in any of the groups, but presented independently.

`type` Character string (required). Type of the field being an R class. The argument determines default configuration for the field input controllers. Available types are 'character', 'factor', 'integer', 'numeric', 'POSIXct', 'Date' and 'logical'.

`input` Character string or JS function. Type of input used. Available types are 'text', 'number', 'textarea', 'radio', 'checkbox' and 'select'. It can also be a JS function which returns the HTML of the said input, this function takes 2 parameters:

- `rule` - the Rule object
- `input_name` - the name of the input

In order to define it, create the function definition as character string and pass it to js. When skipped, the default input will be used based on the provided `type`.

`values` Vector of possible values. Required for limited selection inputs (e.g. 'radio', 'checkbox', 'select').

`value_separator` Character string. Used the split the provided value when a 'text' input is used with an operator allowing multiple values ('in' for example). When skipped, the provided input will be used as a bare value. Needs to be set, when multiple values needs to be provided for 'text' and 'textarea' inputs.

`default_value` Default operator value.

`input_event` (advanced) Character string ('change' by default). Space separated list of DOM events which the builder should listen to detect value changes.

`size` Integer. Only for 'text' and 'textarea' inputs: horizontal size of the input.

`rows` Integer. Only for 'textarea' inputs: vertical size of the input.

`multiple` Logical ('FALSE' default). Set to 'TRUE' if value input controller should accept multiple values. Please make sure the corresponding operators allow to take multiple values to make it work, see mapOperator.

`placeholder` Character string Only for 'text' and 'textarea' inputs: placeholder to display inside the input.

`vertical` Logical (FALSE default). Only for 'radio' and 'checkbox' inputs: display inputs vertically not horizontally.

`validation` List of options for rule validation. See `vignette("validation")`.

`operators` Character vector of operators types to use for this filter. When skipped the filter will use all applicable operators. See listMappedOperators.

`default_operator` Character string. Name of the operator that should be used by default when defining new rules. When skipped the first value from `operators` is used.

`plugin`, `plugin_config` (advanced) Name of a jQuery plugin to apply on the input and plugin configuration. See https://querybuilder.js.org/demo.html#widgets.

`data` List. Additional data that will be added to the returned query - 'input' element returned by queryBuilderInput. Use this to store any functional data you need.

`valueSetter` (advanced) JS function used to set the input(s) value. If provided the default function is not run. The function takes 2 parameters:

- `rule` - the Rule object
- `value`

In order to define it, create the function definition as character string and pass it to js.

`valueGetter` (advanced) function Function used to get the input(s) value. If provided the default function is not run. It takes 1 parameter:

- `rule` - the Rule object

In order to define it, create the function definition as character string and pass it to js.

`unique` Allow to use the filter only once. Can be 'FALSE/TRUE' or 'group' to allow using filter once per group. In order to make it work please enable 'unique-filter' plugin ('plugin = list("unique-filter")') for queryBuilderInput.

Value

A list object storing the filter created filter configuration.

See Also

queryRule

Examples

```
ui <- shiny::fluidPage(
  queryBuilderInput(
    "qb",
    filters = list(
      queryFilter(
        "Species", type = "character", operators = c("in", "equal"),
      values = levels(iris$Species), multiple = TRUE, input = "text", value_separator = ";"
      ),
      queryFilter(
        "Sepal.Length", type = "numeric", values = range(iris$Sepal.Length),
        validation = list(min = 4.3, max = 7.9, step = 0.1)
      )
    )
  ),
  shiny::verbatimTextOutput("expr")
)
server <- function(input, output, session) {}

if (interactive()) {
  shiny::runApp(ui, server)
}
```

`r_to_js_opt_type` *Convert R class to a valid operator JS type*

Description

Convert R class to a valid operator JS type

Usage

```
r_to_js_opt_type(apply_to)
```

Arguments

`apply_to`	Character value - R class to be converted.

Index