

# Package: queryBuilder (via r-universe)

October 25, 2024

**Type** Package

**Title** Programmatic Way to Construct Complex Filtering Queries

**Version** 0.1.0

**Maintainer** Krystian Igras <krystian8207@gmail.com>

**Description** Syntax for defining complex filtering expressions in a programmatic way. A filtering query, built as a nested list configuration, can be easily stored in other formats like 'YAML' or 'JSON'. What's more, it's possible to convert such configuration to a valid expression that can be applied to popular 'dplyr' package operations.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** utils, magrittr, rlang, dplyr, glue, purrr

**Collate** 'queryBuilder-package.R' 'utils.R' 'operators.R'  
'conditions.R' 'config.R' 'rules\_and\_groups.R'  
'parse\_results.R'

**RoxygenNote** 7.3.1

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Krystian Igras [aut, cre], Damien Sorel [cph] (Syntax for defining queries using rules and groups as included in 'jQuery-QueryBuilder' JavaScript framework.)

**Date/Publication** 2024-09-24 19:10:02 UTC

**Repository** <https://krystian8207.r-universe.dev>

**RemoteUrl** <https://github.com/cran/queryBuilder>

**RemoteRef** HEAD

**RemoteSha** 2565ed531494f22bd967bf3740a4687aae7ef3d4

## Contents

queryBuilder-package . . . . .	2
attach_to_list . . . . .	2
err_msg . . . . .	3
in_range . . . . .	3
in_string . . . . .	4
is_empty . . . . .	4
lget_attr . . . . .	5
operator_name_prefix . . . . .	5
prefix_operators_name . . . . .	6
query-condition . . . . .	6
query-operator . . . . .	7
query-rules . . . . .	9
queryBuilderConfig . . . . .	10
queryBuilderConfigClass . . . . .	10
queryToExpr . . . . .	12
query_to_expr_bare . . . . .	13
remove_by_name . . . . .	14
rule_to_expr . . . . .	14
substitute_q . . . . .	15

<b>Index</b>	<b>16</b>
--------------	-----------

---

queryBuilder-package    *Build Filtering Query from Configuration*

---

### Description

Build Filtering Query from Configuration

---

attach\_to\_list            *Combine two lists*

---

### Description

Combine two lists

### Usage

```
attach_to_list(base_list, extra_list)
```

### Arguments

base_list	List to attach objects to.
extra_list	List from which elements should be attached to base_list. Duplicated objects are overwritten.

**Value**

List.

---

err_msg	<i>Generate error message</i>
---------	-------------------------------

---

**Description**

Generate error message

**Usage**

```
err_msg(msg, ..., .envir = parent.frame())
```

**Arguments**

msg	Character string interpreted by <a href="#">glue</a> .
...	Extra arguments passed to <a href="#">glue</a> .
.envir	Environment to evaluate each expression in, passed to <a href="#">glue</a> .

**Value**

Executed error with interpolated message.

---

in_range	<i>Check if value fits to a range</i>
----------	---------------------------------------

---

**Description**

Check if value fits to a range

**Usage**

```
in_range(x, range)
```

**Arguments**

x	Numeric value.
range	Vector of length 2, storing range change limits.

**Value**

A logical vector indicating which elements of x fit into the specified range.

---

in_string	<i>Check if character value matches the provided pattern</i>
-----------	--

---

**Description**

Check if character value matches the provided pattern

**Usage**

```
in_string(x, pattern, ...)
```

**Arguments**

x	String value.
pattern	Pattern that should be matched to x.
...	Extra arguments passed to <a href="#">grepl</a> .

**Value**

A logical vector indicating which elements of x are matching the provided pattern.

---

is_empty	<i>Compare the string to empty value</i>
----------	--

---

**Description**

Compare the string to empty value

**Usage**

```
is_empty(x)
```

**Arguments**

x	String value.
---	---------------

**Value**

A logical vector indicating which elements equal "".

---

lget_attr	<i>Extract attribute of each element from a set of lists</i>
-----------	--

---

**Description**

Extract attribute of each element from a set of lists

**Usage**

```
lget_attr(list_obj, attribute)
```

**Arguments**

list_obj	List of lists. Each nested list should contain el_name object.
attribute	Name of the attribute to extract from each object.

**Value**

Vector of the same length, storing extracted attributes.

---

operator_name_prefix	<i>Prefix used for renaming operators names</i>
----------------------	---

---

**Description**

Required due to erroneous operations on objects with names such as 'in' or 'for'.

**Usage**

```
operator_name_prefix
```

**Format**

An object of class character of length 1.

---

prefix\_operators\_name *Rename operators with the provided prefix*

---

### Description

Rename operators with the provided prefix

### Usage

```
prefix_operators_name(operators)
```

### Arguments

operators      List storing [queryOperators](#).

---

query-condition      *Register new or list existing query conditions*

---

### Description

Condition is two-argument function such as '!' or '&' used to combine pair of rules.

### Usage

```
queryCondition(method)
```

```
setQueryConditions(..., .queryBuilderConfig = queryBuilderConfig)
```

```
listQueryConditions(.queryBuilderConfig = queryBuilderConfig, print = TRUE)
```

```
default_conditions
```

### Arguments

method      R function of two parameters that is used to combine a pair of rules.

...      Name-value pairs defining condition name and method respectively. Should be defined with usage of queryCondition function.

.queryBuilderConfig      R6 class object storing query configuration. See [queryBuilderConfigClass](#).

print      Should the list of operators be printed into console?

### Format

An object of class list of length 2.

**Details**

- `queryCondition`: defines condition method.
- `setQueryConditions`: is used to register the defined conditions in the default or custom `queryBuilderConfigClass` object.
- `listQueryConditions`: returns list of registered conditions.
- `default_conditions`: an object storing default definitions for conditions.

**Examples**

```
setQueryConditions(
  "XOR" = queryCondition(xor)
)
query <- queryGroup(
  condition = "XOR",
  queryRule("am", "equal", 1),
  queryRule("vs", "equal", 1)
)
queryToExpr(query)
```

---

query-operator

*Register new or list existing query operators*

---

**Description**

Operator are functions of maximum two arguments. The first argument is interpreted as a field (e.g. column name), the second one as a filtering value interpreted by operator accordingly. Some operators, such as 'is\_empty' (that compares field values to empty string) don't require any value provided.

**Usage**

```
queryOperator(method)
```

```
setQueryOperators(..., .queryBuilderConfig = queryBuilderConfig)
```

```
listQueryOperators(.queryBuilderConfig = queryBuilderConfig, print = TRUE)
```

```
default_operators
```

**Arguments**

method	R function the operator should be transformed to when parsing result to R expression. The function should take at most two parameters. The first one (obligatory) is variable vector, the second one additional parameters interpreted by operator. Could be negated with exclamation mark e.g. <code>queryOperator(!startsWith)</code> which will be interpreted as the negation of the associated expression.
--------	---

... Name-value pairs defining operator name and method respectively. Should be defined with usage of `queryOperator` function.

`.queryBuilderConfig` R6 class object storing query configuration. See [queryBuilderConfigClass](#).

`print` Should the list of operators be printed into console?

### Format

An object of class `list` of length 20.

### Details

Operators are stored as [quotes](#), that are further interpreted while converting the query to filtering expression.

- `queryOperator`: defines a custom operator that can be used in generated query.
- `setQueryOperators`: is used to register the defined operators in the default or custom [queryBuilderConfigClass](#) object.
- `listQueryOperators`: allows to list available operators for the specific column type.
- `default_operators`: an object storing default definitions for operators.

### Value

A single ‘quote’ storing the provided method.

### Examples

```
listQueryOperators()

in_closed_range <- function(x, range) {
  x >= range[1] & x <= range[2]
}

setQueryOperators(
  "within" = queryOperator(in_closed_range),
  "not_within" = queryOperator(!in_closed_range)
)

query <- queryGroup(
  condition = "AND",
  queryRule("am", "equal", 1),
  queryRule("qsec", "within", c(10, 15)),
  queryRule("disp", "not_within", c(10, 15))
)

queryToExpr(query)
```



---

query-rules	<i>Define filtering query</i>
-------------	-------------------------------

---

## Description

Query is configuration consisting of rules and group. Rule defines a single filtering expression whereas group is combining multiple rules (or nested groups) with the provided condition.

## Usage

```
queryGroup(..., condition = "AND")

queryRule(field, operator, value = NULL, ...)
```

## Arguments

...	Rules defined with queryRule function.
condition	Group condition. By default 'AND' and 'OR' are available. To set custom one use <a href="#">setQueryConditions</a> .
field	Field of the filter applied to the rule. To set custom one use <a href="#">setQueryOperators</a> .
operator	Name of the operator to be applied to the rule.
value	(optional) Values that should be applied to the rule. Some operators, such as 'is_null', don't require any value provided.

## Details

Having the example expression 'a == 1 | (vs == 0 & qsec > 10)' we can distinct the following rules and groups:

Rules: - 'am == 1' - related to 'am' field, applies '==' operator with '1' value, - 'vs == 0' - related to 'vs' field, applies '==' operator with '0' value, - 'qsec > 10' - related to 'qsec' field, applies '>' operator with '10' value.

Groups: - '(vs == 0 & qsec > 10)' - combines two rules ('vs == 0' and 'qsec > 10') with '&' condition, - 'a == 1 | (vs == 0 & qsec > 10)' - combines rule 'a == 1' and group '(vs == 0 & qsec > 10)' with '|' condition.

Such query can be defined by 'queryBuilder' the following way:

```
queryGroup( condition = "OR", queryRule("am", "equal", 1) queryGroup( condition = "AND",
queryRule("vs", "equal", 0), queryRule("qsec", "greater", 10) ) )
```

Connection between conditions and operators names and their R-based counterparts are defined with [queryBuilderConfig](#) class.

The defined query can be then converted to filtering expression with [queryToExpr](#) function.

## Value

Nested lists structure.

**Examples**

```

queryGroup(
  condition = "OR",
  queryRule("am", "equal", 1),
  queryGroup(
    condition = "AND",
    queryRule("vs", "equal", 0),
    queryRule("qsec", "greater", 10)
  )
)

```

---

queryBuilderConfig     *Default object storing 'queryBuilder' configuration.*

---

**Description**

Default object storing 'queryBuilder' configuration.

**Usage**

queryBuilderConfig

**Format**

An object of class queryBuilderConfig (inherits from R6) of length 8.

---

queryBuilderConfigClass  
*R6 class representing 'queryBuilderConfig' object.*

---

**Description**

R6 class representing 'queryBuilderConfig' object.

R6 class representing 'queryBuilderConfig' object.

**Details**

The object is responsible for storing definitions for operators and conditions that are used to generate query expression. It also allows to manage its objects by the provided methods.

**Value**

R6 Class constructor for query configuration (operators, conditions and methods for managing the objects).

**Methods****Public methods:**

- `queryBuilderConfigClass$new()`
- `queryBuilderConfigClass$add()`
- `queryBuilderConfigClass$remove()`
- `queryBuilderConfigClass$get_from_private()`
- `queryBuilderConfigClass$set_to_private()`
- `queryBuilderConfigClass$reset()`
- `queryBuilderConfigClass$clone()`

**Method** `new()`: Create `queryBuilderConfig` object with initialized conditions and operators.

*Usage:*

```
queryBuilderConfigClass$new(  
  conditions = default_conditions,  
  operators = default_operators,  
  ...  
)
```

*Arguments:*

`conditions` Conditions.

`operators` Operators.

... Unused.

*Returns:* The object of class 'queryBuilderConfig'.

**Method** `add()`: Add conditions and conditions to 'queryBuilderConfig' object.

*Usage:*

```
queryBuilderConfigClass$add(conditions = NULL, operators = NULL)
```

*Arguments:*

`conditions` Conditions.

`operators` Operators.

**Method** `remove()`: Remove conditions or operators from 'queryBuilderConfig' object.

*Usage:*

```
queryBuilderConfigClass$remove(conditions_id = NULL, operators_id = NULL)
```

*Arguments:*

`conditions_id` Id of conditions to remove.

`operators_id` Id of operators to remove.

**Method** `get_from_private()`: Get private elements from 'queryBuilderConfig' object.

*Usage:*

```
queryBuilderConfigClass$get_from_private(name)
```

*Arguments:*

`name` Name of the element to get.

**Method** `set_to_private()`: Set private elements to 'queryBuilderConfig' object.

*Usage:*

```
queryBuilderConfigClass$set_to_private(name, value)
```

*Arguments:*

name Name of the element to set.

value New element value.

**Method** `reset()`: Restore default conditions and conditions of 'queryBuilderConfig' object and clear out remaining private objects.

*Usage:*

```
queryBuilderConfigClass$reset()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
queryBuilderConfigClass$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

queryToExpr

*Parse query rules to R filtering expression*

---

## Description

The function takes a list of condition rules provided by the widget (`input[[<widget-name>]]`) and returns valid R expression that can be used for example in [filter](#) function.

## Usage

```
queryToExpr(query, keep_na = FALSE, .queryBuilderConfig = queryBuilderConfig)
```

## Arguments

query Query definition (see [queryRule](#) and [queryGroup](#)).

keep\_na Should query keep or exclude missing values?

.queryBuilderConfig

R6 class object storing query configuration. See [queryBuilderConfigClass](#).

## Value

Object of class 'call'. A filtering expression that can be passed to 'dplyr'-based filtering methods.

## Examples

```
query <- queryGroup(
  condition = "AND",
  queryGroup(
    queryRule(
      field = "Species",
      operator = "equal",
      value = "setosa"
    ),
    queryRule(
      field = "Petal.Length",
      operator = "less",
      value = 1.2
    )
  )
)
queryToExpr(query)
dplyr::filter(iris, !!queryToExpr(query))
```

---

query\_to\_expr\_bare      *Convert query definition to expression*

---

## Description

Convert query definition to expression

## Usage

```
query_to_expr_bare(query, operators, conditions, keep_na)
```

## Arguments

query	Query definition (see <a href="#">queryRule</a> and <a href="#">queryGroup</a> ).
operators	List storing <a href="#">queryOperators</a> .
conditions	List storing <a href="#">queryConditions</a> .
keep_na	Should each rule expression be extended with rule excluding/including 'NA' values?

## Value

Character value storing expression to be parsed.

---

remove_by_name	<i>Remove list elements by their names</i>
----------------	--

---

**Description**

Remove list elements by their names

**Usage**

```
remove_by_name(list_obj, ids)
```

**Arguments**

list_obj	List object.
ids	Objects names to be removed.

**Value**

List.

---

rule_to_expr	<i>Convert rule definition to expression</i>
--------------	--

---

**Description**

Convert rule definition to expression

**Usage**

```
rule_to_expr(rule, operators, keep_na = FALSE)
```

**Arguments**

rule	Rule definition (see <a href="#">queryRule</a> ).
operators	List storing <a href="#">queryOperators</a> .
keep_na	Should the expression be extended with rule excluding/including 'NA' values?

**Value**

Character value storing expression to be parsed.

---

substitute_q	<i>Substitute expression stored as a variable</i>
--------------	---

---

**Description**

Substitute expression stored as a variable

**Usage**

```
substitute_q(x, env)
```

**Arguments**

x	Expression to be substituted.
env	List of arguments to substitute for x.

# Index

## \* datasets

- operator\_name\_prefix, 5
- query-condition, 6
- query-operator, 7
- queryBuilderConfig, 10

attach\_to\_list, 2

default\_conditions (query-condition), 6  
default\_operators (query-operator), 7

err\_msg, 3

filter, 12

glue, 3

grepl, 4

in\_range, 3

in\_string, 4

is\_empty, 4

lget\_attr, 5

listQueryConditions (query-condition), 6

listQueryOperators (query-operator), 7

operator\_name\_prefix, 5

prefix\_operators\_name, 6

query-condition, 6

query-operator, 7

query-rules, 9

query\_to\_expr\_bare, 13

queryBuilder-package, 2

queryBuilderConfig, 9, 10

queryBuilderConfigClass, 6–8, 10, 12

queryCondition, 13

queryCondition (query-condition), 6

queryGroup, 12, 13

queryGroup (query-rules), 9

queryOperator, 6, 13, 14

queryOperator (query-operator), 7

queryRule, 12–14

queryRule (query-rules), 9

queryToExpr, 9, 12

quote, 8

remove\_by\_name, 14

rule\_to\_expr, 14

setQueryConditions, 9

setQueryConditions (query-condition), 6

setQueryOperators, 9

setQueryOperators (query-operator), 7

substitute\_q, 15