

# Package ‘rejustify’

April 6, 2020

**Type** Package

**Title** Support for Rejustify API

**Version** 1.1.1

**Author** M. Wolski <marcin@rejustify.com>

**Maintainer** M. Wolski <marcin@rejustify.com>

**Description** Set of routines to facilitate communication with rejustify API.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Imports** htr (>= 1.4),  
jsonlite (>= 1.6)

**RoxygenNote** 7.0.2

## R topics documented:

adjust	1
analyze	3
callCurl	5
fill	6
isMissing	8
labels	8
register	9
setCurl	10
vis	10
<b>Index</b>	<b>12</b>

---

adjust

*changes the elements of basic blocks used by rejustify API*

---

## Description

The purpose of the function is to provide a possibly seamless way of adjusting blocks used in communication with rejustify API, in particular with the fill endpoint. The blocks include: data structure (structure), default values (default) and matching keys (kets). Items may only be deleted for specific matching dimensions proposed by keys, for the two other blocks it is possible only to change the relevant values.

Upon changes in structure, the corresponding p\_class or p\_data will be set to -1. This is the way to inform API that the original structure has changed and, if learn option is enabled, the new values will be used to train the algorithms in the back end. If learn is disabled, information will not be stored by the API but the changes will be recognized in the current API call.

## Usage

```
adjust(block, column = NULL, id = NULL, items = NULL)
```

## Arguments

block	A data structure to be changed. Currently supported structures include structure, default and keys.
column	The data column (or row in case of horizontal datasets) to be adjusted. Vector values are supported.
id	The identifier of the specific element to be changed. Currently it should be only used in structure with multi-line headers (see analyze for details).
items	Specific items to be changed with the new values to be assigned. If the values are set to NA, NULL or "", the specific item will be removed from the block (only for keys). Items may be multi-valued.

## Value

adjusted structure of the df data set

## Examples

```
#API setup
setCurl()

#register token/email
register(token = "YOUR_TOKEN", email = "YOUR_EMAIL")

#sample data set
df <- data.frame(year = c("2009", "2010", "2011"),
                 country = c("Poland", "Poland", "Poland"),
                 `gross domestic product` = c(NA, NA, NA),
                 check.names = FALSE, stringsAsFactors = FALSE)

#endpoint analyze
st <- analyze(df)

#adjust structures
st <- adjust(st, id = 2, items = list('feature' = 'country'))
st <- adjust(st, column = 3, items = list('provider' = 'IMF', 'table' = 'WEO'))

#endpoint fill
```

```
df1 <- fill(df, st)

#adjust default values
default <- adjust(df1$default, column = 3, items = list('Time Dimension' = '2013') )

#adjust keys
keys <- adjust(df1$keys, column = 3, items = list('id.x' = c(3,1,2) , 'id.y' = c(1,2,3) ) )
keys <- adjust(df1$keys, column = 3, items = list('id.x' = 3 , 'id.y' = NA ) )
```

---

analyze

*communicates with rejustify/analyze API endpoint*


---

## Description

The function submits the data set to the analyze API endpoint and returns the proposed structure of the data. At the current stage data set must be rectangular, either vertical or horizontal.

API recognizes the multi-dimension and multi-line headers. The first inits rows/columns are collapsed using sep character. Make sure that the separator doesn't appear in the header values. It is possible to separate dimensions in single-line headers (see examples below).

The classification algorithms are applied to the values in the rows/columns if they are not empty, and to the headers if the rows/columns are empty. For efficiency reasons only a sample of values in each column is analyzed. To improve the classification accuracy, you can ask the API to draw a larger sample by setting fast=FALSE. For empty columns the API returns the proposed resources that appear to fit well in the empty spaces given the header information and overall structure of df.

The basic properties are characterized by classes. Currently, the API distinguishes between 6 classes: general, geography, unit, time, sector, number. They describe the basic characteristics of the values, and they are further used to propose the best transformations and matching methods for data reconciliation. Classes are further supported by features, which determine the characteristics in greater detail, such as class geography may be further supported by feature country.

Cleaner contains the basic set of transformations applied to each value in a dimension to retrieve machine-readable representation. For instance, values y1999, y2000, ..., clearly correspond to years, however, they will be processed much faster if stripped from the initial y character, such as ^y. Cleaner allows basic regular expressions.

Finally, format corresponds to the format of the values, and it is particularly useful for time-series operations. Format allows the standard date formats (see ?as.Date).

The classification algorithm can be substantially improved by allowing it to recall how it was used in the past and how well it performed. Parameter learn controls this feature, however, by default it is disabled. The information stored by rejustify is tailored to each user individually and it can substantially increase the usability of the API. For instance, the proposed provider for empty row/column with header 'gross domestic product' is IMF. Selecting another provider, for instance AMECO, will teach the algorithm that for this combination of headers and rows/columns AMECO is the preferred provider, such that the next time API is called, there will be higher chance of AMECO to be picked by default. To enable learning option in all API calls by default, run setCurl(learn=TRUE).

If learn=TRUE, the information stored by rejustify include (i) the information changed by the user with respect to assigned class, feature, cleaner and format, (ii) resources determined by provider, table and headers of df, (iii) hand-picked matching values for value-selection. The information will be stored only upon a change of values within groups (i-iii).

**Usage**

```
analyze(
  df,
  shape = "vertical",
  inits = 1,
  fast = TRUE,
  sep = ",",
  learn = getOption("rejustify.learn"),
  token = getOption("rejustify.token"),
  email = getOption("rejustify.email"),
  url = getOption("rejustify.mainUrl")
)
```

**Arguments**

df	The data set to be analyzed. Must be matrix-convertible. If data frame, the dimension names will be taken as the row/column names. If matrix, the row/column names will be ignored, and the header will be set from matrix values in line with inits and sep specification.
shape	It informs the API whether the data set should be read by columns (vertical) or by rows (horizontal). The default is vertical.
inits	It informs the API how many initial rows (or columns in horizontal data), correspond to the header description. The default is inits=1.
fast	Informs the API on how big a sample draw of original data should be. The larger the sample, the more precise but overall slower the algorithm. Under the fast = TRUE the API samples 5 fast = FALSE option it is 25%. Default is fast=TRUE.
sep	The header can also be described by single field values, separated by a given character separator, for instance 'GDP, Austria, 1999'. The option informs the API which separator should be used to split the initial header string into corresponding dimensions. The default is sep=', '.
learn	It should be set as TRUE if the user accepts rejustify to track her/his activity to enhance the performance of the AI algorithms (it is not enabled by default). To change this option for all API calls run setCurl(learn=TRUE).
token	API token. By default read from global variables.
email	E-mail address for the account. By default read from global variables.
url	API url. By default read from global variables.

**Value**

structure of the df data set

**Examples**

```
#API setup
setCurl()

#register token/email
register(token = "YOUR_TOKEN", email = "YOUR_EMAIL")

#sample data set
df <- data.frame(year = c("2009", "2010", "2011"),
```

```
country = c("Poland", "Poland", "Poland"),
`gross domestic product` = NA,
check.names = FALSE, stringsAsFactors = FALSE)
analyze(df)

#data set with one-line multi-dimension header (semi-colon separated)
df <- data.frame(country = c("Poland", "Poland", "Poland"),
  `gross domestic product;2009` = NA,
  `gross domestic product;2010` = NA,
  check.names = FALSE, stringsAsFactors = FALSE)
analyze(df, sep = ";")

#data set with multi-line header
df <- cbind(c(NA, "country", "Poland", "Poland", "Poland"),
  c("gross domestic product", "2009", NA, NA, NA),
  c("gross domestic product", "2010", NA, NA, NA))
analyze(df, inits = 2)
```

---

callCurl

*a wrapper of the httr GET/POST functions*

---

## Description

The function offers a basic functionality of commands GET/POST from httr package to communicate with the APIs. If needed, the proxy settings can be given explicitly, or set in global variables `rejustify.proxyUrl` and `rejustify.proxyPort`.

## Usage

```
callCurl(
  method = "GET",
  url = NULL,
  body = NULL,
  proxyUrl = getOption("rejustify.proxyUrl"),
  proxyPort = getOption("rejustify.proxyPort")
)
```

## Arguments

<code>method</code>	Method of the call (GET or POST).
<code>url</code>	Address of the API.
<code>body</code>	Request body in case of using POST method.
<code>proxyUrl</code>	Url address of the proxy server.
<code>proxyPort</code>	Communication port of the proxy server.

## Value

API response or errors

---

 fill

*communicates with rejustify/fill API endpoint*


---

### Description

The function submits the request to the API fill endpoint to retrieve the desired extra data points. At the current stage dataset must be rectangular, and structure should be in the shape proposed analyze function. The minimum required by the endpoint is the data set and the corresponding structure. You can browse the available resources at <https://rejustify.com/repos>). Other features, including private resources and models, are taken as defined for the account.

The API defines the submitted data set as  $x$  and any server-side data set as  $y$ . The corresponding structures are marked with the same principle, as `structure.x` and `structure.y`, for instance. The principle rule of any data manipulation is to never change data  $x$  (except for missing values), but only adjust  $y$ .

### Usage

```
fill(
  df,
  structure,
  keys = NULL,
  default = NULL,
  shape = "vertical",
  inits = 1,
  sep = ",",
  learn = getOption("rejustify.learn"),
  accu = 0.75,
  form = "full",
  token = getOption("rejustify.token"),
  email = getOption("rejustify.email"),
  url = getOption("rejustify.mainUrl")
)
```

### Arguments

- |                        |   |
|------------------------|---|
| <code>df</code>        | The data set to be analyzed. Must be matrix-convertible. If data frame, the dimension names will be taken as the row/column names. If matrix, the row/column names will be ignored, and the header will be set from matrix values in line with <code>inits</code> and <code>sep</code> specification.   |
| <code>structure</code> | Structure of the $x$ data set, characterizing classes, features, cleaners and formats of the columns/rows, and data provider/tables for empty columns. Perfectly, it should come from analyze endpoint.   |
| <code>keys</code>      | <p>The matching keys and matching methods between dimensions in <math>x</math> and <math>y</math> data sets. The elements in <code>keys</code> are determined based on information provided in data <math>x</math> and <math>y</math>, for each empty column. The details behind both data structures can be visualized by <code>structure.x</code> and <code>structure.y</code>.</p> <p>Matching keys are given consecutively, i.e. the first element in <code>id.x</code> and <code>name.x</code> corresponds to the first element in <code>id.y</code> and <code>name.y</code>, and so on. Dimension</p> |

names are given for better readability of the results, however, they are not necessary for API recognition. `keys` return also data classification in element `class` and the proposed matching method for each part of `id.x` and `id.y`.

Currently, API supports 6 matching methods: `synonym-proximity-matching`, `synonym-matching`, `proximity-matching`, `time-matching`, `exact-matching` and `value-selection`, which are given in a diminishing order of complexity. `synonym-proximity-matching` uses the proximity between the values in data `x` and `y` to the corresponding values in `rejustify` dictionary. If the proximity is above threshold `accu` and there are values in `x` and `y` pointing to the same element in the dictionary, the values will be matched. `synonym-matching` and `proximity-matching` use similar logic of either of the steps described for `synonym-proximity-matching`. `time-matching` aims at standardizing the time values to the same format before matching. For proper functioning it requires an accurate characterization of date format in `structure.x` (`structure.y` is already classified by `rejustify`). `exact-matching` will match two values only if they are identical. `value-selection` is a quasi matching method which for single-valued dimension `x` will return single value from `y`, as suggested by default specification. It is the most efficient matching method for dimensions which do not show any variability.

<code>default</code>	Default values used to lock dimensions in data <code>y</code> which will be not used for matching against data <code>x</code> . Each empty column to be filled, characterized by <code>default\$column.id.x</code> , must contain description of the default values. If missing, the API will propose the default values in line with the history of how it was used in the past.
<code>shape</code>	It informs the API whether the data set should be read by columns ( <code>vertical</code> ) or by rows ( <code>horizontal</code> ). The default is <code>vertical</code> .
<code>inits</code>	It informs the API how many initial rows (or columns in horizontal data), correspond to the header description. The default is <code>inits=1</code> .
<code>sep</code>	The header can also be described by single field values, separated by a given character separator, for instance <code>'GDP, Austria, 1999'</code> . The option informs the API which separator should be used to split the initial header string into corresponding dimensions. The default is <code>sep=', '</code> .
<code>learn</code>	It is <code>TRUE</code> if the user accepts <code>rejustify</code> to track her/his activity to enhance the performance of the AI algorithms (it is not enabled by default). To change this option for all API calls run <code>setCurl(learn=TRUE)</code> .
<code>accu</code>	Acceptable accuracy level on a scale from 0 to 1. It is used in the matching algorithms to determine string similarity. The default is <code>accu=0.75</code> .
<code>form</code>	Requests the data to be returned either in <code>full</code> , or <code>partial</code> shape. The former returns the original data with filled empty columns. The latter returns only the filled columns.
<code>token</code>	API token. By default read from global variables.
<code>email</code>	E-mail address for the account. By default read from global variables.
<code>url</code>	API url. By default read from global variables.

### Value

list consisting of 5 elements: `data`, `structure.x`, `structure.y`, `keys` and `default`

**Examples**

```
#API setup
setCurl()

#register token/email
register(token = "YOUR_TOKEN", email = "YOUR_EMAIL")

#sample data set
df <- data.frame(year = c("2009", "2010", "2011"),
                 country = c("Poland", "Poland", "Poland"),
                 `gross domestic product` = NA,
                 check.names = FALSE, stringsAsFactors = FALSE)

#endpoint analyze
st <- analyze(df)

#endpoint fill
df1 <- fill(df, st)
```

---

isMissing	<i>checks if a variable has non-missing values</i>
-----------	--

---

**Description**

The function checks if a variable is either null, NA, or has an assigned value. In case of vectors, the condition is set on all values.

**Usage**

```
isMissing(x)
```

**Arguments**

x	Variable to test.
---	-------------------

---

labels	<i>communicates with rejustify/getTableLabels API endpoint</i>
--------	--

---

**Description**

The function returns the list of available labels for data set table delivered by provider. If not specified otherwise by dim, the labels are returned for all dimensions. For the list of available provider/tables you can refer to [rejustify.com/repos](http://rejustify.com/repos).

**Usage**

```
labels(
  provider = NULL,
  table = NULL,
  dim = NULL,
  url = getOption("rejustify.mainUrl")
)
```



**Arguments**

provider	Relevant provider (required).
table	Relevant provider (required).
dim	Relevant dimension from provider/table (optional).
url	API url. By default read from global variables.

**Value**

list containing the pairs of value code and label

**Examples**

```
#API setup
setCurl()
labels('IMF', 'WEO')
labels('IMF', 'WEO', 'WEO Country')
```

---

register	<i>sets the token and email as global variables</i>
----------	---

---

**Description**

The function stores the account details into memory to be easier accessed by rejustify package. The email must correspond to the token that was assigned to it. To register an account visit [rejustify.com](http://rejustify.com).

**Usage**

```
register(token = NULL, email = NULL)
```

**Arguments**

token	API token.
email	E-mail address for the account.

**Value**

errors only

**Examples**

```
register(token = "YOUR_TOKEN", email = "YOUR_EMAIL")
```

---

setCurl	<i>sets the default configuration for curl calls</i>
---------	--

---

### Description

The command stores the connection details into memory to be easier accessed by rejustify package.

### Usage

```
setCurl(
  mainUrl = "https://api.rejustify.com",
  proxyUrl = getOption("rejustify.proxyUrl"),
  proxyPort = getOption("rejustify.proxyPort"),
  learn = getOption("rejustify.learn")
)
```

### Arguments

mainUrl	Main address for rejustify API calls. Default is set to <code>https://api.rejustify.com</code> , but depending on the customer needs, the address may change.
proxyUrl	Address of the proxy server.
proxyPort	Port for communication with the proxy server.
learn	Enable AI learning in all API calls by setting <code>learn=TRUE</code> . You can also specify the learn option in the relevant functions directly.

### Examples

```
#setting up connection through proxy
rejustify::setCurl(proxyUrl = "PROXY_ADDRESS", proxyPort = 8080)
```

---

vis	<i>visualizes the matching/filling procedures</i>
-----	---

---

### Description

The function aims at simplyfying.

### Usage

```
vis(object, column = NULL, details = FALSE)
```

### Arguments

object	Object returned by the fill function.
column	The empty column for which we want to visualize the matching/filling procedures. By default the first empty column is taken.
details	Displays id, column, class and feature elements in the table. They are not displayed by default.

**Value**

list containing the pairs of value code and label

**Examples**

```
## Not run:  
#rdf is an object returned by fill function  
vis(rdf)  
vis(rdf, details = TRUE)  
  
## End(Not run)
```

# Index

adjust, 1  
analyze, 3  
callCurl, 5  
fill, 6  
isMissing, 8  
labels, 8  
register, 9  
setCurl, 10  
vis, 10