Package 'shiny.i18n'

October 14, 2022

Title Shiny Applications Internationalization

Version 0.2.0

Description It provides easy internationalization of Shiny applications. It can be used as standalone translation package to translate reports, interactive visualizations or graphical elements as well.

Depends R (>= 3.3.0)

Imports yaml, jsonlite, methods, stringr, R6, glue, shiny, rstudioapi, utils

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/Appsilon/shiny.i18n

BugReports https://github.com/Appsilon/shiny.i18n/issues

RoxygenNote 7.1.1

Suggests knitr, googleLanguageR, rmarkdown, testthat, covr

NeedsCompilation no

Author Dominik Krzemiński [cre, aut], Krystian Igras [aut], Appsilon [cph]

Maintainer Dominik Krzemiński <dominik@appsilon.com>

Repository CRAN

Date/Publication 2020-10-02 14:00:02 UTC

R topics documented:

`.i18_config` *Global i18n config list.*

Description

Global i18n config list.

Usage

```
.i18_config
```

Format

An object of class `list` of length 3.

`.onLoad` *onLoad*

Description

On package load it updates .i18_config reading yaml file from config.

Usage

```
.onLoad(libname, pkgname)
```

Arguments

libname	library name
pkgname	package name

.translator_options *Translator options*

Description

Translator options

Usage

```
.translator_options
```

Format

An object of class list of length 3.

---

check_value_presence *Check for value presence*

Description

If value is not present in vector it takes its first value.

Usage

```
check_value_presence(val, vect, warn_msg = "")
```

Arguments

val	element of vector vect
vect	vector of values
warn_msg	warning message to be displayed if val not in vect

Value

updated val

column_to_row *Column to row*

Description

Returns the same data.frame where one column is a rowname now.

Usage

```
column_to_row(data, colname)
```

Arguments

data	data.frame with data
colname	character with column name

Value

data.frame with one column less

create_translation_addin *Create translation file addin*

Description

Create translation file addin

Usage

```
create_translation_addin()
```

create_translation_file

Create translation file

Description

Auxiliary shiny.i18n function that searches for all key expressions (eg. surrounded by `i18n$t()` tag in the script).

Usage

```
create_translation_file(path, type = "json", handle = "i18n", output = NULL)
```

Arguments

`path`	character with path of the file that needs to be inspected for key translations
`type`	type of the example output file with translations, either "json" or "csv"
`handle`	name of `Translator` object within script from `path`
`output`	if NULL (default) the output will be saved with a default file name ("translation.json" for JSON and "translate_lang.csv" for CSV)

extract_key_expressions

Extract key expressions

Description

Extract key expressions

Usage

```
extract_key_expressions(text, handle = "i18n")
```

Arguments

`text`	character with text of the script
`handle`	character with Translator object handle (default: 'i18n')

Value

vector of characters with key expressions

get_i18n_config *Get i18n config*

Description

Get i18n config

Usage

```
get_i18n_config(field)
```

Arguments

field	a field from configuration file

Value

character with option from config.yaml file

i18n_state *Create i18n state div*

Description

This hidden div contain information about i18 translation state.

Usage

```
i18n_state(init_language)
```

Arguments

init_language	key language code

Value

shiny tag with div "i18n-state"

`init_i18n` *Creates new i18n Translator object*

Description

Creates new i18n Translator object

Usage

```
init_i18n(
  translation_csvs_path = NULL,
  translation_json_path = NULL,
  translation_csv_config = NULL,
  automatic = FALSE
)
```

Arguments

`translation_csvs_path`	character with path to folder containing csv translation files. See more in Details.
`translation_json_path`	character with path to JSON translation file. See more in Details.
`translation_csv_config`	character with path to configuration file for csv option.
`automatic`	logical flag, indicating if i18n should use an automatic translation API.

Value

i18n object

Examples

```
## Not run:
 i18n <- init_i18n(translation_csvs_path = "../csvdata/")
 i18n <- init_i18n(translation_json_path = "translations.json")

## End(Not run)
```

load_local_config *Load Local YAML Config*

Description

Load Local YAML Config

Usage

```
load_local_config(yaml_config_path)
```

Arguments

yaml_config_path	path to yaml config file

Value

list of config options or empty list if file not exists

multmerge *Mulitple Merge*

Description

Inspired by: https://www.r-bloggers.com/merging-multiple-data-files-into-one-data-frame/

Usage

```
multmerge(filenames, sep = ",")
```

Arguments

filenames	character vector with filenames
sep	fields separator

Value

data.frame with merged files

read_and_merge_csvs *Read and merge CSVs*

Description

This function reads and merges data from multiple csv files in given folder.

Usage

```
read_and_merge_csvs(dir_path, sep = ",")
```

Arguments

dir_path	character with path to directory with csv files
sep	fields separator

Value

data.frame with CSV files content

save_to_csv *Save example i18n file to CSV*

Description

It saves translation data structure with language key code "key" and language to translate name "lang" as an example of a translation csv file.

Usage

```
save_to_csv(key_expressions, output_path = NULL)
```

Arguments

key_expressions	vector with key expression to translate
output_path	character with path to output file (default: "translate_lang.csv" if NULL)

save_to_json *Save example i18n file to json*

Description

It saves translation data structure with language key code "key" as an example of a translation JSON file.

Usage

```
save_to_json(key_expressions, output_path = NULL)
```

Arguments

key_expressions	vector with key expression to translate
output_path	character with path to output file (default: "translation.json" if NULL)

translate_with_google_cloud *This provides functions for automatic translations with online APIs Translate with Google cloud*

Description

This is wrapper for `gl_translate` function from `googleLanguageR` package.

Usage

```
translate_with_google_cloud(txt_to_translate, target_lang)
```

Arguments

txt_to_translate	character with text to translate
target_lang	character with language code

Translator *Translator R6 Class*

Description

Translator R6 Class

Translator R6 Class

Details

This creates shinny.i18n Translator object used for translations. Now you can surround the pieces of the text you want to translate by one of the translate statements (ex.: `Translator$t("translate me")`). Find details in method descriptions below.

Methods

Public methods:

- `Translator$new()`
- `Translator$get_languages()`
- `Translator$get_translations()`
- `Translator$get_key_translation()`
- `Translator$translate()`
- `Translator$t()`
- `Translator$set_translation_language()`
- `Translator$parse_date()`
- `Translator$parse_number()`
- `Translator$automatic_translate()`
- `Translator$at()`
- `Translator$use_js()`
- `Translator$clone()`

Method `new()`**:** Initialize the Translator with data

Usage:

```
Translator$new(
  translation_csvs_path = NULL,
  translation_json_path = NULL,
  translation_csv_config = NULL,
  separator_csv = ",",
  automatic = FALSE
)
```

Arguments:

`translation_csvs_path` character with path to folder containing csv translation files. Files must have "translation_" prefix, for example: `translation_<LANG-CODE>.csv`.

translation_json_path character with path to JSON translation file. See more in Details.
translation_csv_config character with path to configuration file for csv option.
separator_csv separator of CSV values (default ",")
automatic logical flag, indicating if i18n should use an automatic translation API.

Method `get_languages()`**:** Get all available languages

Usage:

```
Translator$get_languages()
```

Method `get_translations()`**:** Get whole translation matrix

Usage:

```
Translator$get_translations()
```

Method `get_key_translation()`**:** Get active key translation

Usage:

```
Translator$get_key_translation()
```

Method `translate()`**:** Translates 'keyword' to language specified by 'set_translation_language'

Usage:

```
Translator$translate(keyword, session = shiny::getDefaultReactiveDomain())
```

Arguments:

keyword character or vector of characters with a word or expression to translate
session Shiny server session (default: current reactive domain)

Method `t()`**:** Wrapper to `translate` method.

Usage:

```
Translator$t(keyword, session = shiny::getDefaultReactiveDomain())
```

Arguments:

keyword character or vector of characters with a word or expression to translate
session Shiny server session (default: current reactive domain)

Method `set_translation_language()`**:** Specify language of translation. It must exist in 'languages' field.

Usage:

```
Translator$set_translation_language(transl_language)
```

Arguments:

transl_language character with a translation language code

Method `parse_date()`**:** Parse date to format described in 'cultural_date_format' field in config.

Usage:

```
Translator$parse_date(date)
```

Arguments:

date date object to format

Method `parse_number()`**:** Numbers parser. Not implemented yet.

Usage:

```
Translator$parse_number(number)
```

Arguments:

`number` numeric or character with number

Returns: character with number formatting

Method `automatic_translate()`**:** Translates 'keyword' to language specified by 'set_translation_language' using cloud service 'api'. You need to set API settings first.

Usage:

```
Translator$automatic_translate(keyword, api = "google")
```

Arguments:

`keyword` character or vector of characters with a word or expression to translate

`api` character with the name of the API you want to use. Currently supported: `google`.

Method `at()`**:** Wrapper to `automatic_translate` method

Usage:

```
Translator$at(keyword, api = "google")
```

Arguments:

`keyword` character or vector of characters with a word or expression to translate

`api` character with the name of the API you want to use. Currently supported: `google`.

Method `use_js()`**:** Call to wrap translation in span object. Used for browser-side translations.

Usage:

```
Translator$use_js()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
  i18n <- Translator$new(translation_json_path = "translation.json") # translation file
  i18n$set_translation_language("it")
  i18n$t("This text will be translated to Italian")

## End(Not run)

# Shiny example
if (interactive()) {
library(shiny)
library(shiny.i18n)
```

```
 #to run this example  make sure that you have a translation file
 #in the same path
i18n <- Translator$new(translation_json_path = "examples/data/translation.json")
i18n$set_translation_language("pl")
ui <- fluidPage(
  h2(i18n$t("Hello Shiny!"))
)
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
}
```

`update_lang` *Update language (i18n) in UI*

Description

It sends a message to session object to update the language in UI elements.

Usage

```
update_lang(session, language)
```

Arguments

`session`	Shiny server session
`language`	character with language code

See Also

usei18n

`usei18n` *Use i18n in UI*

Description

This is an auxiliary function needed to monitor the state of the UI for live language translations.

Usage

```
usei18n(translator)
```

Arguments

`translator`	shiny.i18 Translator object

Examples

```
if (interactive()) {
  library(shiny)
  library(shiny.i18n)
  # for this example to run make sure that you have a translation file
  # in the same path
  i18n <- Translator$new(translation_json_path = "translation.json")
  i18n$set_translation_language("en")

  ui <- fluidPage(
    usei18n(i18n),
    actionButton("go", "GO!"),
    h2(i18n$t("Hello Shiny!"))
  )

  server <- shinyServer(function(input, output, session) {
    observeEvent(input$go,{
      update_lang(session, "pl")
    })
  })

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

`validate_names` *Validate Column Names*

Description

Validate if n-th column name of data.frames (given in list) is the same.

Usage

```
validate_names(list_df, n = 1)
```

Arguments

`list_df`	list of data frames
`n`	integer denoting column number

Value

TRUE if names of n-th columns of data.frames is the same, FALSE otherwise.

Index