**Why?**

Have you ever created a multilingual Shiny app? It is very likely that the answer isÂ **no**, because Shiny just doesn't have any good tools for that. In  AppsilonÂ we came across the internationalization problem many times, so we decided to make a tool which makes a live easier when it comes to mulitlingual.Â shiny.i18nÂ is the new kid on the block and still under rapid development, but theÂ 0.1.0Â version is already ready to go.

***NOTE:****shiny.i18nÂ usage is not limited to Shiny apps. You can use it as standalone R package for generating multilingual reports or visualizations. We decided on this name because Shiny is the most common and obvious use-case scenario.*

**Where?**

The latest version the package (0.1.0) has been released recently to CRAN, so you can simply get it like this:

install.packages("shiny.i18n")

In the GitHub page, you can also find more examples, read the documentation, post some issues or your own contributions.

**How?**

Now it's time to learn more about how to useÂ shiny.i18n. The package utilizes specific translation file data formats. Currently two approaches are supported.

**JSON translation format**

Example of a JSON translation file for English and polish language you can find below:

{

"languages": ["en", "pl"],

"translation": [

{"en": "Hello Shiny!", "pl": "Witaj Shiny!"},

{"en": "Number of bins:", "pl": "Liczba podziaÅ‚ek"},

{

"en": "This is description of the plot.",

"pl": "To jest opis obrazka."

},

{

"en": "Histogram of x",

"pl": "Histogram x"

},

{

"en": "Frequency",

"pl": "CzÄ™stotliwoÅ›Ä‡"

}

]

}

It consists of a single fileÂ translation.jsonÂ with two mandatory fields:

* “languages”with a list of all language codes;
* “translation”with a list of dictionaries assigning translation to a language code.

Other fields (such asÂ cultural\_date\_format) are optional and if missing will be read fromÂ default configÂ yamlÂ file as mentioned below.

|  |
| --- |
| cultural\_date\_format: "%d/%m/%Y" |
|  | cultural\_bignumer\_mark: " " |
|  | cultural\_punctuation\_mark: "," |

**CSV translation format**

Another approach is to use a CSV format. The main idea behind it is to support distributed translation tasks among many translators. Instead of having to concatenate results of work from many translators together, we can just store them in a common folder with the specific name of the file:Â translation\_.csv.

You can imagine a situation with the following folder structure:

translations/

translation\_pl.csv

translation\_it.csv

translation\_kl.csv

which have translations forÂ plÂ Polish,Â itÂ Italian and – as the language code is completely arbitrary –Â klÂ for Klingon language.

Let's have a look at how typical CSV translation file should look like:

en, it

Hello Shiny!, Ciao Shiny!

Histogram of x, Istogramma di x

This is description of the plot., Questa Ã¨ la descrizione della trama.

Frequency, Frequenza

Number of bins:, Numero di scomparti:

Change language, Cambia lingua

This time we need to remember that all CSV files from one dictionary must share common reference language in the left column (which isÂ enÂ English in the above case).

**Creating the app**

To integrate our translations with Shiny we start from loading packages and an example JSON file.

library(shiny)

library(shiny.i18n)

i18n <- Translator$new(translation\_json\_path = "translations/translation.json")

Having that, we can check in RStudio console all languages stored inÂ i18nÂ object.

> i18n$languages

[1] "en" "pl"Â "kl"

Now within Shiny app we need to surround all text elements withinÂ i18n$translateÂ or in shortÂ i18n$tÂ method. For instance:

sliderInput("bins",

i18n$t("Number of bins:"),

min = 1,

max = 50,

value = 30)

)

to translate a message displayed byÂ sliderInputÂ element, or:

titlePanel(i18n$t("Hello Shiny!"))

to translate a titlePanel content.  
If we decide to run an instance of the app with a specific language (let’s say Klingon) we should call:

i18n$set\_translation\_language("kl")

right after definingÂ TranslatorÂ object.

Below you can see the full example:

library(shiny)

library(shiny.i18n)

i18n <- Translator$new(translation\_json\_path = "../data/translation.json")

i18n$set\_translation\_language("pl")

ui <- shinyUI(fluidPage(

titlePanel(i18n$t("Hello Shiny!")),

sidebarLayout(

sidebarPanel(

sliderInput("bins",

i18n$t("Number of bins:"),

min = 1,

max = 50,

value = 30)

),

mainPanel(

plotOutput("distPlot"),

p(i18n$t("This is description of the plot."))

)

)

))

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

output$distPlot <- renderPlot({

x <- faithful[, 2]

bins <- seq(min(x), max(x), length.out = input$bins + 1)

hist(x, breaks = bins,

col = "darkgray", border = "white",

main = i18n$t("Histogram of x"), ylab = i18n$t("Frequency"))

})

})

shinyApp(ui = ui, server = server)

Above code should convince you about how easy it is to start using shiny.i18n. For more examples once again I refer you to the GitHub page. We hope that shiny.i18n will help you to forget about problems with internationalization.