# Package: traduire (via r-universe)

August 13, 2024

**Title** Internationalisation on the Fly

**Version** 0.1.0

**Description** Internationalisation ('i18n') support for R, switching languages at runtime, based on the 'i18next' 'JavaScript' library. This provides an alternative approach to internationalisation than supported directly by R.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Imports** V8, R6, glue

**Suggests** jsonlite, knitr, mockery, rmarkdown, testthat

**RoxygenNote** 7.1.1

**VignetteBuilder** knitr

**Language** en-GB

**URL** https://github.com/reside-ic/traduire

**BugReports** https://github.com/reside-ic/traduire/issues

**Repository** https://mrc-ide.r-universe.dev

**RemoteUrl** https://github.com/reside-ic/traduire

**RemoteRef** master

**RemoteSha** 0109a8498df21691989f7e0e85e0049c4cfc89af

## Contents

## i18n *Create translator object*

### Description

Create a new translator object

### Usage

```
i18n(resources, ..., options = NULL)
```

### Arguments

| | |
|---|---|
| `resources` | Path to a json file containing translation resources. If given in this way, then on-demand translation loading (via `resource_pattern`) is disabled unless a currently unexposed i18next option is used. |
| `...` | Named options passed to `traduire_options` |
| `options` | Options object passed to `traduire_options` |

### Warning

Note that the argument list here *will* change. The only part of this that we consider stable is that the first argument will represent a resource bundle.

### Examples

```
path <- system.file("examples/simple.json", package = "traduire")
obj <- traduire::i18n(path)
obj$t("hello", language = "fr")
```

## lint_translations *Lint translations*

### Description

Find likely translation errors using static source code analysis. We look into a series of R source files and identify all calls to `t_` and examine the key and interpolation data used within these calls.

### Usage

```
lint_translations(path, obj, language = NULL, root = NULL, title = NULL)

lint_translations_package(root, language = NULL)
```

### Arguments

`path` Path to the translations to test. This can be a character vector of file names and directory names - directory names are expanded (non-recursively) to find all `.R` files.

`obj` A translator object, or ‘NULL‘. If ‘NULL‘, then the report will indicate only strings that might be translated vs bare strings, rather than check the likely success of translation.

`language` Optional language to use as the language to lint. Each language must be at present linted separately

`root` A root directory, below which the files in `path` will be treated relatively (e.g., pass a package’s root directory here, and then set `path = "R"`). Using this shortens the reported filenames considerably

`title` An optional title that will be used in reporting

### Details

This will expand in future and is currently quite limited. Currently, only calls to `t_()` are inspected - not calls to `$t()` in a translator object, not calls to `$replace` etc. The `package` and `name` arguments to `t_()` are also currently ignored, which will lead to spurious errors. A version of this that automatically works for packages will also be written. The key and interpolation data must (currently) be literals - you cannot save the key or data as a variable and pass that by variable name.

The function `lint_translations_package` is a wrapper around `lint_translations` to make working with a package a bit easier. It's a bit flakey at the moment because the translator will be loaded from the currently installed package, but the usage will be checked in the source tree. This will be addressed in a future version (see reside-90)

### Value

A `lint_translations` object, which can be passed through to `lint_translations_report`

---

## `lint_translations_report` *Show translation lint report*

---

### Description

Report on translation linting. At present this produces a single-page html report that can be opened in a web browser. Future versions will include reporting to the terminal.

### Usage

```
lint_translations_report(x, file = tempfile(fileext = ".html"))
```

### Arguments

`x` A `lint_translations` object from `lint_translations`

`file` File to save the report in (default is to use a temporary file with a file extension of `.html`).

### Value

This function is called for the side effect of viewing a report. It will return (invisibly) the path to the produced report.

## `traduire_options` *Create a set of options for traduire*

### Description

Takes a named set of options and optionally an already created options object. If both are set, named options will overwrite any options in the object.

### Usage

```
traduire_options(..., options = NULL)
```

### Arguments

`...` Named options

language The default language for the translation

default_namespace The default namespace to use. If not given, then `i18next` assumes the namespace `translation`

debug Logical, indicating if i18next's debug output should be turned on. This will result in lots of output via `message` about various i18next actions.

resource_pattern A pattern to use for on-demand loading of translation resources. Only works if `translations` is `NULL` at present.

namespaces A vector of namespaces to load. Namespaces not listed here may not be loaded as expected (use `debug = TRUE` to work out what is going on). The default (`NULL`) will use i18next's logic, which is to use `translation` as the only loaded namespace. This creates some issues if `default_namespace` is set here, as the default namespace will not be loaded. A future version of this package will probably do better with the logic here.

languages A vector of languages to *preload*. You can always add additional languages using the `load_language` method. Note that the adding a language here does not (yet) mean that failure to load the language is an error.

fallback The fallback language to use. The options here are to use a character string (a single fallback to use for all languages), a character vector (a series of languages to use in turn, listed from first to try to last to try) or a named list of language-fallback mappings, e.g., `list("de-CH": c("fr", "it"), "es": "fr")`.

escape Logical, indicating if the translation output should be, by default, escaped (see the i18next interpolation documentation). The i18next implementation is to prevent xss attacks, and so is disabled by default in traduire.

`options` traduire_options object

### Details

These options are passed to i18next.

### Value

A 'traduire_options' object

### Examples

```
opts <- traduire::traduire_options()
opts <- traduire::traduire_options(language = "fr")
opts <- traduire::traduire_options(default_namespace = "tr", options = opts)
```

## `translator_register` *Register a translator*

### Description

Register a translator within the `traduire` package, and use it directly. This will allow your code to access translations via `translator_translate` (or more concisely `t_`) without having to pass around a translation object. If called from package code (and assuming a single translator per package) then the `name` argument can be omitted and will be automatically converted into `package:<packagename>`).

### Usage

```
translator_register(resources, ..., name = NULL)

translator_unregister(name = NULL)

translator_translate(..., name = NULL, package = NULL)

t_(..., name = NULL, package = NULL)

translator(name = NULL, package = NULL)

translator_set_language(language, name = NULL, package = NULL)

translator_list()
```

### Arguments

`resources` Path to a json file containing translation resources. If given in this way, then on-demand translation loading (via `resource_pattern`) is disabled unless a currently unexposed i18next option is used.

`...` For `translator_register`, arguments passed to `traduire_options` to build the translator object. All arguments are accepted. For `translator_translate` and `t_`, arguments passed to the `$t` method of the translator object, being `string`, `data`, `language` etc.

`name` Optional name for the translator. This should be used only when not using this interface from a package (e.g., from a shiny application). If using from a package you can omit both `name` and `package`, and if interacting with translations from another package you should use the `package` argument (see below).

`package` Optional name for the package to find a translator in. This cannot be provided for `translator_register` and `translator_unregister` as these should either be registered by `name` or the package will be determined automatically.

`language` Language to use, passed through to `i18n`'s `set_language` method

### Use in package code

The intention is that this would typically be called from `.onLoad`, something like:

```
.onLoad <- function(...) {
  path <- system.file("traduire.json", package = "hello", mustWork = TRUE)
  traduire::translator_register(path, "en")
}
```

and then used from that package's code as

```
traduire::t_("key")
```

The language option for this translator can be changed by

```
traduire::translator_set_language("es")
```

Every package's translator object is isolated from every other package, and if the `traduire` functions are called from your package code, then the correct translator should be found automatically.

If you need to get a translation for another package, you should use `package` argument, for example:

```
traduire::t_("key", package = "other")
```

You can change the language in another package (e.g., using `traduire::change_language("en", package = "other")`) but should be careful to reset this using the returned reset function.

It is not possible to unregister a translator in another package, or to overwrite one.

Translators provided in other packages will be listed by `traduire::translator_list` with the prefix `package:` (e.g., `package:other`) however, you should not access them directly using `name = "package:other"`.

## Examples

```
path <- system.file("examples/simple.json", package = "traduire")
traduire::translator_register(path, name = "myexample")
traduire::t_("hello", language = "fr", name = "myexample")
"myexample" %in% traduire::translator_list()
traduire::translator_unregister("myexample")
```

# Index