Translator

Manual

Easy and simple translations for words and expressions

transl 0.0.1 MIT

Maycon F. Melo[*]

QUICK START

```
#import "@preview/transl:0.0.1": transl
#transl(data: yaml("lang.yaml"))

// Get "I love you" in Spanish:
#set text(lang: "es")
#transl("I love you")

// Translate every "love" to Italian:
#set text(lang: "it")
#show: doc => transl("love", doc)
```

DESCRIPTION

Get comprehensive and contextual translations, with support for regular expressions and *Fluent*[1] localization. This package have one main command, `#transl`, that receives one or more expression strings, searches for each of them in its database and then returns the translation for each one.

The expressions are the text to be translated, they can be simple words or longer text excerpts, or can be used as identifiers to obtain longer text blocks at once. Regular expressions are supported as string patterns — not `#regex` elements.

All the conceptual structure of *transl* and its idea is heavily inspired on the great *linguify*[2] package.

[*]https://github.com/mayconfmelo

[1]https://projectfluent.org/

[2]https://typst.app/universe/package/linguify

OPTIONS

These are all the options and its defaults used:

```
#import "@preview/transl:0.0.1": transl
#transl(
  from: auto,
  to: auto,
  data: none,
  mode: context(),
  args: (:),
  ..expr
)
```

`from:` string

Initial origin language.

`to:` string auto

Final target language — fallback to `#text.lang` when not set.

`data:` yaml dictionary

Translation file — see the `docs/assets/example.yaml` file.

`mode:` type

Type of value returned: an opaque `context()` or a contextualized `str` — used as `#context transl(mode: str)`.

`args:` dictionary

Fluent arguments used to customize the translation output.

`..expr` strings

Expressions to be translated.

FLUENT DATA

```
#import "@preview/transl:0.0.1": fluent
#fluent(
  ..data,
  lang: (),
  args: (:)
)
```

Fluent is a localization solution (L10n) developed by Mozilla that can easily solve those wild language-specific variations that are tricky to fix in code, like gramatical case, gender, number, tense, aspect, or mood. When used to set `#transl(data)`, this function signalizes *transl* to use its embed Fluent plugin instead of the standard mechanism (YAML). It may need to be wrapped in an `#eval` to work properly.

`..data` string

Path to where the `ftl` files are stored in the project (requires `#eval`) or `"file!"` followed by the Fluent code itself (does not require `#eval`).

`lang:` array string

The languages used — each corresponds to *lang.ftl* inside the given path.

`args:` dictionary none

Additional arguments passed to Fluent.

STANDARD DATA

After setting Fluent as localization mechanism, *transl* will use it from now on. To go back to the default localization mechanism the `#std()` command must be used:

```
#import "@preview/transl:0.0.1": std
#std(
  data: (:)
)
```

The `#std` command does not need to be wrapped in an `#eval`.

USE CASES

STANDARD TRANSLATION DATABASE

```
#transl(data: yaml("lang.yaml"))
```

Before any proper translation, it is required to insert some translation data into *transl*, so it will know what to translate. These data imports are cumulative: newer entries overrides the older ones, so its possible to add multiple files (e.g., one for each language) for better organization. See `docs/example/lang.yaml` to learn more about the structure of a standard translation database.

FLUENT TRANSLATION DATABASE

```
#transl(
  data: eval( fluent(data: "path", lang: ("pt", "es")) )
)
```

To enable the support for Fluent localization, is necessary to set the database using `#fluent`, which can resolve the paths to the `flt` files and read them. In the code above, the files `path/pt.ftl` and `path/es.ftl` will be added to the standard translation database. Because of some Typst limitations on `#read`, it is required to wrap it inside an `#eval` command for now; alternatively, passing `"file!"` followed by tye Fluent code itself (as string) gets rid of the evaluation:

```
#transl(
  data: fluent(data: "file!" + read("path/pt.ftl"), lang: "pt")
)
```

Note that this syntax allows to import data for only one language at a time. After set Fluent, all next *transl* command will use it as localization mechanism; to go back to the standard localization mechanism, use:

```
#transl( data: std(yaml("lang.yaml")) )
```

The YAML database can be ommited if there is already a standard translation database registered.

GET TRANSLATION

```
#transl("expression")
```

The *expressions* are simple strings that contains the text to be translated, or regular expression patterns that matches it. If more than one expression is given at the same time, their translations are concatenated with a space in between. When gets an expression, *transl* tries to find it as string then as a regex pattern when nothing is found — in this case, the first entry that matches the regex will be used.

```
#transl("exp.*?n")
```

Fluent databases does not support regex patterns directly because the expressions must be simple text identifiers, like variables.

SET ORIGINAL LANGUAGE

```
#transl("expression", from: "en")
```

Defines the initial language of the expression, before translation. This is an optional feature used to get the expression itself as translation when `#transl(from) == #transl(to)`.

SET TARGET LANGUAGE

```
#transl("expression", to: "pt")
```

Defines the language of the translation obtained. This is an optional feature that fallback to the current `#text.lang` when not set.

USING SHOW RULES

```
#show: doc => transl("expression", doc)
```

When used as a `show` rule, *transl* allows to automatically translate all the expressions found in the text without using the command `#transl` each time. When multiple expression values are given, each one of them is translated through the text; and when no expression is given, all available database entries for the language selected are used — refer to `docs/example/main.typ` for an exemple.

SHOW RULES WITH FLUENT

There is some limitations when using Fluent in `show` rules as it does not support to retrieve data using regex or text expressions, it must be a simple identifier (like a variable). However, this can be achieved using both Fluent and standard databases: by storing the localized translation in Fluent database and the expression to be searched in standard database.

```
#set text(lang: "es")
#transl(data: yaml("en.yaml"))
#transl(data: eval( fluent(data: "path", lang: "es") ))

#show: doc => transl("identifier", from: "en", doc)
```

The `show` rule above translates `identifier` to Spanish using Fluent, but tries to find in standard database the expression to be searched in the text. Thus, the same identifier “`love`” could retrieve the translation “te amo mucho” from Fluent and also the expression “i love you so much” from standard database, so that in practice it translates “I love you so much” as “te amo mucho”. This also allows to use regex patterns in these situations.

GET CONTEXTUAL STRING

```
#context transl("expression", mode: str)
```

This allows to manage and tweak the translated string received from *transl* without the barrier of the `context()` value: when using contextual data, *transl* returns an opaque `context()` value that cannot be manipulated; but this mode returns the contextualized string value instead, used inside a `#context` block.

This is useful for package mantainers that need to manipulate or use the translated value in elements that only allows string arguments, like `#raw()`.

CONTEXT-FREE TRANSLATIONS

```
#transl("expression", to: "pt", data: yaml("lang.yaml"))
```

To completely get rid of all `context()` is required to set `#transl(..expr, to, data)` in the same command, this way all needed information is available at once and nothing needs to be contextually retrieved. This returns a simple `string` without the need of any additional `#context` blocks.

COPYRIGHT

The logo was obtained from Flaticon website.

The Fluent support is a fork of a *linguify*[3] feature, and all the overall project concept is heavily inspired in this great package.

[3]https://typst.app/universe/package/linguify