# Translator

Manual

Easy and simple translations for words and expressions

transl 0.1.0 MIT

Maycon F. Melo[*]

## QUICK START

```
#import "@preview/transl:0.1.0": 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 expression patterns are supported when *transl* is used in `show` rules.

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 by *min-book*:

```
#import "@preview/transl:0.1.0": transl
#transl(
  from: none,
  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 by *transl*: a `context()` or a `str` — the latter requires `#context transl(mode: string)`.

**`args:`** dictionary

Fluent arguments used to customize the translation output.

**`..expr`** strings

Expressions to be translated.

## FLUENT DATA

```
#import "@preview/transl:0.1.0": fluent
#fluent(
  path,
  lang: (),
  args: (:)
)
```

Fluent is a localization solution (i10n) 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 needs to be wrapped in an `eval` to work properly.

**`path`** string

The path where the `ftl` files are stored in the project.

**`lang:`** array string

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

**`args:`** dictionary none

Additional arguments passed to Fluent.

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:0.1.0": std
#std(
  data: (:)
)
```

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

## USE CASES

### STANDARD TRANSLATION DATABASE

```
#import "@preview/transl:0.1.0": transl
#transl(data: yaml("lang.yaml"))
```

Before any proper translation, 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 the `docs/assets/example.yaml` file to learn more about the structure of a translation database.

### FLUENT TRANSLATION DATABASE

```
#import "@preview/transl:0.1.0": transl, fluent
#transl(
  data: eval( fluent(data: "path", lang: ("pt", "es")) )
)
```

To enable the support for Fluent localization, is necessary to set the database using `#fluent`, that resolves the paths to all the `flt` files and reads them. In the code above, the files `path/pt.ftl` and `path/es.ftl` will be added to the translation database. Because of some Typst limitations on `#read`, it is required to wrap it inside an `#eval` command for now.

After set to Fluent, every *transl* command from now on will use it as localization mechanism, to go back to the standard localization mechanism, use:

```
#import "@preview/transl:0.1.0": transl, std
#transl( data: std(yaml("lang.yaml")) )
```

The YAML database can be ommited if there os already a standard translation database registerd.

## GET TRANSLATION

```
#import "@preview/transl:0.1.0": transl
#transl("expression")
```

The *expressions* are simple strings that contains the text to be translated. If more than one expression is given at tue same time, its translations are returned together, separated by space.

## SET ORIGINAL LANGUAGE

```
#import "@preview/transl:0.1.0": transl
#transl("expression", from: "en")
```

Defines the initial language of the expression, before translation. This is an optional feature used to get tye expression itself as translation when `#transl(from) == #text.lang`. The expression can be a single word or an text excerpt.

## SET TARGET LANGUAGE

```
#import "@preview/transl:0.1.0": transl
#transl("expression", to: "pt")
```

Defines the language of the translation obtained for of the expression given. This is an optional feature that fallback to the current `#text.lang` when no set.

## USING SHOW RULES

```
#import "@preview/transl:0.1.0": transl
#show: doc => transl("expression", doc)
```

This way of using *transl* allows to automatically translate the expressions found in the text, without using the command `#transl` each time. When multiple expression values are given, each one of them is replaced for its translation in tue text; and when no expression is given, all available entries in the database for the language selected are used. Tue expressions can also be regular expression patterns (`string`) — though in this case the translation in the database must be idenficied by the regex pattern instead of the text that would match it — see the `docs/assets/example.yaml` for more information.

### GET CONTEXTUAL STRING

```
#import "@preview/transl:0.1.0": transl
#context transl("expression", mode: str)
```

This allows to manage and tweak the translated string received from *transl* without the barrier of `context()`, but to use it is required to wrap all the code that modifies the translated string in a `context` block. This is useful for package mantainers that need to manipulate or to use the translated value in elements that only allows `string` arguments, like `#raw()`.

### CONTEXT-FREE TRANSLATIONS

```
#import "@preview/transl:0.1.0": transl
#transl("expression", to: "pt", data: yaml("lang.yaml"))
```

To completely get rid of all `context()` is required to set `#transl(..expr, to, data)` in tue 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