



Translator

Manual

Easy and simple translations for words and expressions

typst

transl

0.1.0

MIT

Maycon F. Melo*

Contents

Quick Start	2
Description	2
Options	2
Fluent Data	3
Standard Data	3
Use Cases	4
Set Standard Translation Database	4
Set Fluent Translation Database	4
Get Translation	4
Set Original Language	5
Set Target Language	5
Using Show Rules	5
Show Rules With Fluent	5
Fluent Arguments	6
Get Contextual String	6
Context-Free Translations	7
Copyright	7

*<https://github.com/mayconfmelo>

Quick Start

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

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

#set text(lang: "it")
// Translate every "love" to Italian:
#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 by the great *linguify*² package.

Options

```
#transl(
  from: auto,
  to: auto,
  data: none,
  mode: context(),
  ..expr
)
```

from: string
Initial origin language.

¹<https://projectfluent.org/>

²<https://typst.app/universe/package/linguify>

to: string autoFinal target language — fallback to `#text.lang` if not set.**data:** yaml fluent dictionaryTranslation file (see `docs/assets/example.yaml` file).**mode:** context() str

Type of value returned: an opaque context or string.

..expr strings

Expressions to be translated.

Fluent Data

```
#fluent(
  ..data,
  lang: (),
)
```

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 grammatical case, gender, number, tense, aspect, or mood. When used to set `#transl(data)`, this function signals *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 "path/" "file!data"

Fluent data: a path to where the `ftl` files are stored (requires `#eval`), or the data itself preceded by `"file!"` (does not require `#eval`).

lang: array string

Languages of the Fluent data: multiple corresponding each to a file inside the data path, or a single one corresponding to the data itself passed.

Standard Data

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

```
#std(
  data: (:)
)
```

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

Use Cases

Set 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.

Set 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 `ftl` 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 the 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 omitted 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 example.

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.

Fluent Arguments

Fluent allows to use special arguments to obtain gramatical contextualization, like different plural and singular sentences, cardinal numbering, or just insert custom expressions inside the translation. When using a Fluent database, any additional argument in `#transl` is considered a Fluent argument. For example:

```
#transl("identifier", case: "nominative")
#transl("identifier", gender: "masculine")
#transl("identifier", number: "singular")
#transl("identifier", tense: "present")
#transl("identifier", name: "John Doe")
```

None of these are `#transl` arguments, therefore they are passed to Fluent. Be aware that these same additional arguments will throw a `#panic` when using the standard database though.

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 maintainers 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

Copyright © 2025 Maycon F. Melo.

This manual is licensed under MIT.

The manual source code is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

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>