Translator

Manual

Easy and simple translations for words and expressions

---

typst transl 0.2.0 MIT

Maycon F. Melo*

Contents

*https://github.com/mayconfmelo

Quick Start

```
#import "@preview/transl:0.2.0": transl
#transl(data: yaml("lang.yaml"))

#set text(lang: "es")
#transl("I love you")

#show: transl.with("love")
```

Description

Get comprehensive and localized translations, with support for regular expressions and *Fluent*[1] files. This package comes with only one `#transl` command used to both set the translation database and retrieve translations.

The expressions are the texts to be translated; they can be words, phrases, or regular expression strings. Multiple expressions can be used in a single command, where each one will be retrieved and concatenated (separated by space).

Although written from scratch, the package's conceptual structure is heavily inpired by the *linguify*[2] package; the Fluent WASM plugin was also borrowed from this great package.

Options

```
#transl(
  from: auto,
  to: auto,
  data: (:),
  mode: context(),
  ..expr
)
```

`from:` string

Initial origin language.

`to:` string

Final target language; set to `#text.lang` when `auto`.

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

`data:` string yaml dictionary

Translation file; `string` sets Fluent database and `dictionary` sets standard database (see Translation database section).

`mode:` context() str

How to retrieve values: as opaque context or string values.

`..expr` arguments

Expressions to be translated (pos) and localization arguments (named).

User Guide

Translation database

The `#transl` searches and retrieve translation data from a database that must be set before its use; the database available is always the one set earlier in code. The translation database is split in 2 to store standard and Fluent data separately.

Standard database

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

The standard database is a simple storage based on dictionaries: all data is stored in the following structure:

```
l10n: std

lang:
  expression: Translation
```

`l10n:` std

Database type; `std` defines a standard database.

`lang:` dictionary

Language code.

`expression:` string

Expression to be retrieved by `#transl`.

Fluent database

```
#transl(data: read("en.ftl"), lang: "en")
#transl(data: yaml("ftl.yaml"))
```

The Fluent database stores files used by Fluent in translation and localization. There are two ways to insert data in a Fluent database: `#read` each FTL file with its `lang` individually; or set the whole YAML database with all files inside. A Fluent file follows the structure:

```
identifier = Translation
```

While a YAML database that contains multiple files follows the structure:

```
l10n: ftl

lang: |
  identifier = Translation
```

`l10n:` std

Database type; `ftl` defines a Fluent database.

`lang:` string

Language code.

`identifier:` string

Fluent identifier used as expression by `#transl`.

Using FTL files allow to store data for each language separately, while requires each file to be imported manually with a `lang` option; and using a YAML database requires all Fluent data to be in the same file, while allows to set the entire database using one command. Also note that one is loaded by `#read`, while the other by `#yaml`.

Translation

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

After setting the database, just use *expressions*. The expression is the text to be translated, searched in the database to return the proper translation. If multiple expressions are used in the same command, all translations are concatenated with spaces in between. An expression can also be a regular expression, used to match one expression from the database.

```
#transl("expr.*n?", "expression", "expression")
```

For each expression, `#transl` searches it in the standard database; if not found, tries to match `regex(expression)` in this database; and if not found, search in the Fluent database.

Since Fluent files uses identifiers instead of expressions, regular expressions and expressions with spaces and special characters are not supported.

Original Language

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

Defines the initial language of the expression, before translation. Allows to return the expression itself when both `#transl(from, to)` options are equal.

Target Language

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

Defines the language of the translation obtained. Fallback to current `#text.lang` if not set.

Translate all expression occurrences

```
#show: transl.with("expression")
```

When used in a `#show` rule, `#transl` automatically substitutes all occurrences for each expression that follows it. If no expression is given, tries to retrieve and use all expressions available in database for the current language.

To use expressions and Fluent data in `#show` rules, just store the rule expression pattern in the standard database, under the same identifier but in `#transl(from)` language.

Localization arguments

```
#transl(arg: "Value")
```

Any arguments other than the default ones (see Options section) are treated as additional localization arguments, they select localization cases (Fluent only) and replace special `{{$arg}}` patterns in translated values retrieved.

Typst

```
#transl(
  "response",
  answer: "no",
  name: "John",
)
```

Fluent

```
response = { $answer ->
  [yes] I do know {{ $name }}
  [no] I don't know {{ $name }}
  [maybe] I don't remember any {{ $name }}
}
```

Ways to retrieve translations

Opaque context

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

The translation database is stored in a `#state`, and therefore requires `#context` to be accessed; this option provides the context needed to access it, but makes `#transl` return opaque `context()` values that cannot be modified later. Used as default when no `#transl(mode)` is set.

Context-dependent string

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

The translation database is stored in a `#state`, and therefore requires `#context` to be accessed; this option returns the translation without context provided, this allows to access and modify the value but requires to manually set a `#context` in code.

Plain string

```
#transl(to: "es", data: yaml("std.yaml"), "expression")
```

When both, database and target language, are directly known from the command arguments, the translation is performed without requiring any context.

The `#transl(data)` value used is not set to the database `#state`, and therefore will not be available in later command uses.

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