linguify manual

Abstract

linguify is a package for loading strings for different languages easily.

Version: 0.4.2
Authors: jomaway + community contributions
License: MIT

Contents

This manual shows a short example for the usage of the *linguify* package inside your document. If you want to **include linguify into your package** make sure to read the section for package authors.

Usage

Basic Example

Load language data file: ⟶ See database section for content of `lang.toml`

```
#set-database(toml("lang.toml"))
```

Example input:

```
#set text(lang: "LANG")
#smallcaps(linguify("abstract"))
=== #linguify("title")

Test: #linguify("test")
```

Lang	Output
en	Abstract **A simple linguify example** Test: testing
de	Zusammenfassung **Ein einfaches Linguify Beispiel** Test: testen
es	Resumen **Un ejemplo sencillo de linguify** Test: testing ***Info***: *The key «test» is missing in the «es» language section, but as we specified a default-lang in the* `conf` *it will display the entry inside the specified language section, which is «en» in our case.* *To* ***disable*** *this behavior delete the* `default-lang` *entry from the* `lang.toml` *.*
cz	Abstract **A simple linguify example** Test: testing ***Info***: *As the lang data does not contain a section for "cz" this entire output will fallback to the default-lang.* *To* ***disable*** *this behavior delete the* `default-lang` *entry from the* `lang.toml` *.*

Database

The content of the `lang.toml` file, used in the example above looks like this.

```
[conf]
default-lang = "en"

[lang.en]
title = "A simple linguify example"
abstract = "Abstract"
test = "testing"

[lang.de]
title = "Ein einfaches Linguify Beispiel"
abstract = "Zusammenfassung"
test = "testen"

[lang.es]
title = "Un ejemplo sencillo de linguify"
abstract = "Resumen"

[lang.fr]
title = "Un exemple simple de linguify"
abstract = "résumé"
```

Handling outlines

Linguify looks up translations in the *currently active* database. This is a problem with outlines, because outline entries are rendered at a different location than the elements they refer to (headings, figures). In this case, an unexpected database may be used.

If you use multiple databases for different parts of your document, and also use translations for your heading or figure captions, you can add the following show rule to ensure correct database lookup:

```
#show outline.entry: it => database-at(it.element.location(), it)
```

See the `database-at()` function for more details.

Alternatively, you can use `linguify-raw()` to manually provide context to *linguify*. By doing so, you can control that *linguify* is using the context from outside the heading, which means the same context in the heading and the outline:

```
// wrong: `linguify` provides its own context inside the heading
// (unless you use `database-at` as shown above)
= #linguify("...")

// correct: use externally provided context that refers to outside the heading
#context [= #linguify-raw("...")]
```

This will make *all* context match between heading and outline (e.g. also the current language), whereas using `database-at()` only affects the database being used.

Information for package authors.

As the database is stored in a typst state, it can be overwritten. This leads to the following problem. If you use *linguify* inside your package and use the `set-database()` function it will probably work like you expect. But if a user imports your package and uses *linguify* for their own document as well, he will overwrite the your database by using `set-database()`. Therefore it is recommend to use the `from` argument in the `linguify()` function to specify your database directly.

Example:

```
// Load data
#let lang-data = toml("lang.toml")

// Useage
#linguify("key", from: lang-data)
```

This makes sure the end user still can use the global database provided by *linguify* with `set-database()` and calling.

⟶ Have a look at the gentle-clues package for a real live example.

Fluent support

Thanks to sjfhsjfh, *linguify* also has Fluent[1] support. Fluent allows for more complex localization, such as accounting for separate plural or other counting forms. To use Fluent, the `conf.data-type` key of your database needs to be set to `"ftl"`. In addition, each language contains a Fluent language definition instead of many keys for all the terms. A complete example of a Fluent database could look like this:

```
[conf]
default-lang = "en"
# set database type to Fluent
data-type = "ftl"

# add arguments available to Fluent translations by default
[ftl.args]
name = "Lore"

[lang]
# each language is a single key containing a whole Fluent file
en = '''
title = A linguify example - with Fluent
abstract = Abstract
hello = Hello, {$name}!
heading = {$headingCount ->
    [one] {$headingCount} heading
   *[other] {$headingCount} headings
}
'''

de = '''
title = Ein linguify Beispiel - mit Fluent
abstract = Zusammenfassung
hello = Hallo, {$name}!
heading = {$headingCount ->
    [0] keine Überschriften
    [one] eine Überschrift
   *[other] {$headingCount} Überschriften
}
'''
```

Since embedding one file inside another is not optimal for things like IDE support, *linguify* also has `load-ftl-data()` to load languages from separate files. Heres a simple example of how to load

[1]Project Fluent

translations from Fluent files, which are kept in `l10n` directory and named with the language code, e.g. `en.ftl` and `de.ftl`.

```
// my-document.typ
#import "@preview/linguify:0.4.2": *
// Define the languages you have files for.
#set-database(eval(load-ftl-data("./l10n", ("en", "de"))))
```

Note how there is a call to `eval()`, since the *linguify* package can't read your translation files directly; instead *linguify* only generates the code that does the reading and lets you execute it.

Likewise, you have to maintain the language list used in database initialization since Typst currently does not list files in a directory. Of course, you can use an external file to store the list of language files and use that to load the ftl files. One option is to use the TOML database file for this:

Store config inside a `lang.toml` file.

```
[conf]
default-lang = "en"
data-type = "ftl"

[ftl]
languages = ["en", "de"]
path = "./l10n"

# no `[lang]`, it will be populated
# by the code on the right
```

Load config inside your document.

```
#let data = toml("lang.toml")

// insert ftl files into database
#(data.lang = data.ftl.languages.map(lang => {
  (lang, read(path + "/" + lang + ".ftl"))
}).to-dict())

#set-database(data)
```

The code above is roughly equivalent to what the `load-ftl-data()` function does, except it lets you store the list of languages in the data file and sets the `default-lang`.

Contributing

If you would like to integrate a new i18n solution into *linguify*, you can set the `conf.data-type` described in the database section. And then add implementation in the `get-text()` function for your data type.

Reference

database-at

Temporarily overrides the location at which the translation database is looked up. This is typically used to change the lookup inside outlines. Consider this:

```
#set-database(toml("a.toml"))
#outline()
= linguify("foo")

#set-database(toml("b.toml"))
= linguify("bar")
```

In this example, the `foo` translation should be loaded from `a.toml` and `bar` from `b.toml`. However, the outline is covered by `a.toml` – including the entry for the `bar` heading!

Adding the following show rule at the beginning fixes this:

```
#show outline.entry: it => database-at(it.element.location(), it)
```

This will make linguify look up the translations for each outline entry at the location the referenced element (heading) is located.

Parameters

```
database-at(
  loc,
  body
)
```

set-database

Set the default linguify database

The data must contain at least a lang section like described at `database`.

Parameters

```
set-database(data: dictionary ) -> content (state-update)
```

data `dictionary`

the database which will be set to `database`

reset-database

Clear current database

Parameters

```
reset-database() -> content (state-update)
```

get-text

Get a value from a L10n data dictionary. If the key does not exist, `none` is returned.

Parameters

```
get-text(
  src: dictionary ,
  key: string ,
  lang: string ,
  mode: string ,
  args
)
```

src `dictionary`

The dictionary to get the value from.

key `string`

The key to get the value for.

lang `string`

The language to get the value for.

mode `string`

The data structure of src

Default: `"dict"`

linguify-raw

fetch a string in the required language. must have a context beforehand to access the global database/lang

Parameters

```
linguify-raw(
  key: string ,
  from: dictionary ,
  lang: string ,
  default: any ,
  args
) -> content
```

key `string`

The key at which to retrieve the item.

from `dictionary`

database to fetch the item from. If auto linguify’s global database will used.

Default: `auto`

lang `string`

the language to look for, if auto use `context text.lang` (default)

Default: `auto`

default `any`

A default value to return if the key is not part of the database.

Default: `auto`

linguify

fetch a string in the required language. provides context for `linguify-raw` function.

Parameters

```
linguify(
  key: string ,
  from: dictionary ,
  lang: string ,
  default: any ,
  args
) -> content
```

key `string`

The key at which to retrieve the item.

from `dictionary`

database to fetch the item from. If auto linguify’s global database will used.

Default: `auto`

lang `string`

the language to look for, if auto use `context text.lang` (default)

Default: `auto`

default `any`

A default value to return if the key is not part of the database.

Default: `auto`

database

None or dictionary of the following structure:

- `conf`
  - `data-type` (string): The type of data structure used for the database. If not specified, it defaults to `dict` structure.
  - `default-lang` (string): The default language to use as a fallback if the key in the preferred language is not found.
  - …
- `lang`
  - `en` : The English language section.
  - …

location-stack

A stack (array) of `location` s to use instead of `here()` when looking up the current database. This is used internally to support looking up translations in e.g. outlines relative to a heading's or figure's location, instead of the outline's.

When the stack is empty, the current location is used.

get-message
Returns the message from the ftl file

Parameters

```
get-message(
  source: string ,
  msg-id: string ,
  args: dictionary ,
  default: string
) -> string
```

source `string`

the content of the ftl file

msg-id `string`

the identifier of the message

args `dictionary`

the arguments to pass to the message

Default: `none`

default `string`

the default value to return if the message is not found

Default: `none`

load-ftl-data

Constructs the data dict needed in `linguify.typ`

Returns a `str`, use `eval` to convert it to a dict

Example:

```
eval(load-ftl-data("path/to/ftl", ("en", "fr")))
```

Parameters

```
load-ftl-data(
  path: string ,
  languages: array
) -> string
```

path `string`

the path to the directory containing the ftl files

languages `array`

the list of languages to load