ngx-i18nsupport

0.17.1 • Public • Published a year ago

Readme

Explore BETA

6 Dependencies

3 Dependents

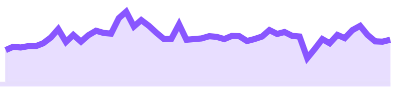
41 Versions

Install

```
npm i ngx-i18nsupport
```

Weekly Downloads

12,393



Version	License
0.17.1	MIT
Unpacked Size	Total Files
879 kB	117

Issues

12

Pull Requests

5

Homepage

github.com/martinroob/ngx-i18nsupport#readme

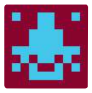
Repository

github.com/martinroob/ngx-i18nsupport

Last publish

a year ago

Collaborators



>_ Try on RunKit

Report a vulnerability

build passing dependencies out of date devDependencies out of date coverage 86% npm package 0.17.1

ngx-i18nsupport

Some tooling to be used for Angular i18n workflows.

> This page contains just a very short description about the installation process and usage. For details have a look at the **Tutorial for using xliffmerge** contained in the Wiki pages (recently updated to Angular 6).

> There is also some support to use dynamic translations based on `ngx-translate` in parallel. See details at the Wiki page **ngx translate usage**

Table of Contents

Introduction

Angular has a specific way of dealing with internationalization (i18n). It is described in the official documentation Angular Cookbook **Internationalization (i18n)**.

Said in one sentence,

- markup your strings to translate in your templates with an attribute `i18n`
- run the Amgular extraction tool (`ng-xi18n`) to extract the strings in an XML Format called [XLIFF-1.2]
- copy and then translate the extracted file for every language you plan to support
- run the ng compiler to generate a special version of your app for the different languages

There is an excellent Blog Article by Phillippe Martin **Deploying an i18n Angular app with angular-cli** , which describes it in detail.

But there are some maior gaps in the workflow. That´s where this tool comes into play.

First, you have to create a complete translation, otherwise, the ng compiler will not generate a version. It is not possible to run with **partial translation**.

Second, whenever you change something in your app, you have to regenerate the xliff, but there is no documented way how to **merge** this with the already **existing** translated files. There are new translation unit, that you have to merge in, and there are translation units, that do not exist any more.

Installation

`npm install -g ngx-i18nsupport`

This will install a script called `xliffmerge`.

You can then integrate the script in your angular i18n workflow, typically in the `package.json` script section:

```
[

-i18n": "ng xi18n --output-path i18n && xliffmerge --profile xliffmerge
```

Usage

```
xliffmerge [options] [language*]
```

Merge translations from a generated master to language specific files

Options:

```
    <json-configurationfile> read confguration data from profile (see be
    activate debug mode (produce more output)
    quiet mode, only errors and warnings are show
    output usage information
    output the version number
```

`language` is an ISO shortcut for the language(s) you use, e.g. "en", "de", "de-ch", ...

`json-configurationfile` is a json file containing some configuration values used by `xliffmerge`. Most times you simply mention it in your npm script using the `--profile`-option. Instead of having a separate file you can also put the configuration in your `package.json`. If no profile is given, `xliffmerge` looks for configuration values in `package.json` (starting with version 0.14).

The following list shows all allowed content (every value is optional, there is a json schema available too):

```
{
  "xliffmergeOptions": {
    "srcDir": "i18n",
    "genDir": "i18n",
    "i18nFile": "messages.xlf",
    "i18nBaseFile": "messages",
    "i18nFormat": "xlf",
    "encoding": "UTF-8",
    "defaultLanguage": "en",
    "languages": ["en", "de"],
    "removeUnusedIds": true,
    "supportNgxTranslate": false,
    "ngxTranslateExtractionPattern": "@@|ngx-translate",
    "useSourceAsTarget": true,
    "targetPraefix": "",
    "targetSuffix": "",
    "beautifyOutput": false,
    "allowIdChange": false,
    "autotranslate": false,
    "apikey": "",
    "apikeyfile": "",
    "verbose": false,
    "quiet": false
  }
}
```

The options are:

- `srcDir` (string, default "."): directory, where the master file is expected
- `genDir` (string, default "."): directory, where files are written to (normally identical with srcDir)
- `i18nFile` (string, default "messages.xlf"): master file (relativ to srcDir)

- `i18nBaseFile` (since 0.13) (string, default "messages"): base name of master file (without suffix) (relativ to srcDir). Only used if `i18nFile` is not set. Useful to generate different filename for different projects. E.g. `i18bBaseFile: project1messages` produces language files `project1messages.de.xlf` ...
- `i18nFormat` (string, default "xlf"): `xlf` for XLIFF 1.2 or `xlf2` for XLIFF 2.0 or `xmb` for XML Message Bundles
- `encoding` (string, default "UTF-8"): expected encoding of xlf, xlf2 or xmb files
- `defaultLanguage` (string, default "en"): the native language used in your templates
- `languages` (array of strings): list of languages (if not spefified at command line)
- `removeUnusedIds` (boolean, default `true`): flag, if unused IDs should be removed during merge
- `supportNgxTranslate` (boolean, default `false`): flag to active json translation files for ngx-translate
- `ngxTranslateExtractionPattern` (string, default `@@|ngx-translate`): defines what messages are exported to json translation files for ngx-translate. For details how to use it have a look at the Wiki Page ngx translate usage.
- `useSourceAsTarget` (boolean, default `true`): flag, if source should be copied to target for new trans-units
- `targetPraefix` (since 0.12.0) (string, default """): when the flag `useSourceAsTarget` is set and a source is copied to target, then the target string will be praefixed by this value. E.g. `targetPraefix: "%%"` and source contains the string "menu", target will contain "%%menu" at the end.
- `targetSuffix` (since 0.12.0) (string, default """): when the flag `useSourceAsTarget` is set and a source is copied to target, then the target string will be suffixed by this value. E.g. `targetSuffix: "%%"` and source contains the string "menu", target will contain "menu%%" at the end.
- `beautifyOutput` (since 0.16.0) (boolean, default `false`): when set to true, the generated xml output will be send through a beautifier (pretty-data) to get consistent output. See (xliffmerge #64 Could xlf output be better formatted) for details.
- `allowIdChange` (since 0.11.0) (boolean, default `false`): flag, wether xliffmerge should merge transunits with changed IDs. When there is only a small change in the original message text, e.g. a trailing white space, the Angular extraction tool will change the ID of the unit. This means that the translations of the unit are lost. When you activate this flag, `xliffmerge` will check for units with only white space changes and merge them correctly.
- `autotranslate` (since v0.7.0) (boolean or array of strings, default `false`): flag, if new units should be automatically translated by Google Translate. You can also specify a string

array with the languages you want to auto translate (e.g `"autotranslate": ["fr", "ru"]`). For details how to use it have a look at the Wiki Page xliffmerge-autotranslate-feature.

- `apikey` (since v0.7.0) (string, default ""): API key for usage of Google Translate. This or `apikeyfile` or env var `API_KEY_FILE` must be set if you activate autotranslate by setting `autotranslate`.
- `apikeyfile` (since v0.7.1) (string, default ""): file that contains API key for usage of Google Translate. This or `apikey` or env var `API_KEY_FILE` must be set if you activate autotranslate by setting `autotranslate`.
- `verbose` (boolean, default `false`): controls output
- `quiet` (boolean, default `false`): controls output

Generate (untranslated) language files, if not already there

When you run `xliffmerge`, it will read the master xliff file **messages.xlf**. This is the file generated by the Angular extraction tool `ng-xi18n`.

Then for every language you specified, it will create a new language specific file, e.g **messages.en.xlf** or **messages.en.xlf**.

These files are mainly copies of the master, but they contain the target translations for all translation units of the master.

If the master contains

```
<trans-unit id="xy...">
  <source>Hello, world</source>
</trans-unit>
```

then the generated file for English (messages.en.xlf) will contain

```
<trans-unit id="xy...">
  <source>Hello, world</source>
  <target state="final">Hello, world</target>
</trans-unit>
```

and the generated file for German (messages.de.xlf) will contain

```
<translation-unit>
  <source id="xy..">Hello, world</source>
  <target state="new">Hello, world</target>
</translation-unit>
```

Obviously this is not the correct translation, it is just no translation. This is shown by the **state** `new`. The next step you have to do is to translate the file (or to let it translate). Depending on the software you use for translation you can filter for that state `new`.

> Have a look at my sister project TinyTranslator. It can filter for new untranslated entries and allows to edit xlf file very easily.

The file for English on the other hand is correct. So, due to the fact, that English is the **default language** here, the state is `translated`.

The Angular compiler can now use both files to generate language specific versions.

Merge new translation units into the existing language files

Generating translation files for each language and translating them is just the beginning.

When you continue developing your app, you will make changes on the existing templates, add new one, etc.

Now, when you are ready to publish a new release, you will run the `ng-xi18n` tool again and it will create a new `messages.xlf` for you. There will be new trans-units in it, and there will be trans-units, that are now removed from the file.

But what do you do with your existing translation for the old version of your app? You don`t want to restart translating the whole stuff.

xliffmerge can do it for you. It will merge the newly created messages.xlf into your existing language specific files.

Whenever it finds an ID in `messages.xlf`, that does not exist in the language file, it will create it with a dummy translation and mark it as `new`, just the same way, that happens when creating a new language file.

Whenever it finds the ID in the language file, it will not touch it, so you will not lose the translation.

And whenever it finds an ID in the language file, that does not exist in the `messages.xlf` anymore, it will remove it from the language file (you can prevent that by setting `removeUnusedIds` to `false` in the profile).

So after running it, you just have to translate the new parts.

> Once again: TinyTranslator might help you to do that.

Tests

`npm test`

This will run a testsuite that checks all relevant aspects of xliffmerge.

Contributing

I did not really think about contributions, because it is just a small experimental project.

But if you are interesting, send me an email, so that we can discuss it.

References

- Angular Cookbook Internationalization (i18n)
- Phillippe Martin Deploying an i18n Angular app with angular-cli
- Roland Oldengarm: Angular 2: Automated i18n workflow using gulp
- XLIFF 1.2 Specification: XLIFF-1.2
- XLIFF 2.0 Specification: XLIFF-2.0
- My Tiny Translator Tool: TinyTranslator

Keywords

i18n tooling angular xliff xmb

Help

Documentation

Community

Resources

Advisories

Status

Contact

About

Company

Blog

Careers

Webinars

Press

Newsletter

Terms & Policies

Policies

Terms of Use

Code of Conduct

Privacy