Tutorial for using xliffmerge with angular cli

Jump to bottom

Martin Roob edited this page on 7 Jan 2019 · 25 revisions

(Updated to Angular 6, 7 and schematics support)

This tutorial shows in a few step, how to use `xliffmerge` in an Angular App, which is generated with `angular-cli`.

It shows the basic workflow for preparing an application for translation, translating it, running it, changing it, translating it again, running it again...

Start a new project

If you haven't done it at all, install `angular-cli` globally:

```
npm install -g @angular/cli@latest
```

Then you can create a new project:

```
ng new sampleapp
```

```
cd sampleapp
```

Prepare the app for translations

For this tutorial we assume that your preferred language is not English, but something else, e.g. German. So you want to write your application in German and translate it to English later. This is not a hard requirement, of course you can use English as the default. I just decided to do it this way, because it is the way I do it.

So as a first step we change our application to German and additionally mark all text, that needs translation, with an `i18n`-Attribute. For details have a look at the Angular Cookbook about Internationalization.

Our `src/app/app.component.html` (generated by the cli) now looks like this:

```
<div style="text-align:center">
  <h1 i18n>
    Willkommen zu {{ title }}!
  </h1>
  <img ..>
</div>
<h2 i18n>Hier einige Links, die den Anfang erleichtern: </h2>
```

Try running it (`ng serve` or `npm run start`, then open `http:\\localhost:4200` in a browser), fix the test cases, if you want (they test for 'app works!' and will fail after the change).

It looks like this (a little bit simplified)

Add the xliffmerge tooling to your project

There is an `ng add Schematic` to add the toling to your new project. Just type

```
ng add @ngx-i18nsupport/tooling --i18nLocale=de --languages de,en
```

The parameter `--i18nLocale=de` tells that you are using German as your default language ("en" is the default if you do not specify it). And the parameter `--languages de,en` sets all the languages that you are planning to use in the moment (of course you can add more languages later on).

You will see some output like this

```
    added script extract-i18n to //package.json
    added npm script to extract i18n message, run "npm run extract-i18n" for
extraction
    added script start-en to //package.json
    added npm script to start app for language en, run "npm run start-en"
    added build configuration en to project sampleapp
    added build configuration for language "en" to project "sampleapp"
    added serve configuration en to project sampleapp
    added serve configuration for language "en" to project "sampleapp"
    added architect builder xliffmerge to project sampleapp
    added builder xliffmerge to project "sampleapp"
UPDATE package.json (1647 bytes)
UPDATE angular.json (4613 bytes)
```

As you can see there are changes on your workspace file `angular.json` and on your `package.json`. There are some new scripts installed in your `package.json` to start your application using different languages and there is a new script to extract translatable parts from your application.

Generate a messages.xlf file

After the initial changes described in the previous chapter your application now contains some i18n marked stuff. So now you can extract this stuff for translation.

angular-cli has a build in task `xi18n` for this. So you could just type

```
ng xi18n sampleapp --output-path i18n --i18n-locale de
```

The `ng add @ngx-i18nsupport/tooling` added a script to your `package.json` that does this for you. You will find the following in your `package.json` scripts section:

```
{
  [...]
  "scripts": {
    [...]
    "extract-i18n": "ng xi18n sampleapp --i18n-format xlf --output-path i18n --
i18n-locale de && ng run sampleapp:xliffmerge"
  }
```

```
  [...]
}
```

Then you can type

```
npm run extract-i18n
```

Some words about the parameters used in the command.

`--output-path` specifies the path (relativ to `src`) where the generated file will be saved. We prefer to have it under `src/i18n`.

`--i18n-locale` specifies the language that is used in the templates (the default language, German in this tutorial).

After running the command you will find a newly generated file `src/i18n/messages.xlf`. If you open it with a text editor, you will see that it is an XML file containing some somewhat strange markup wich is defined by XLF

```
...
      <trans-unit id="91fcc40bc784cacec7c70d5f1eee77ebc1d8d308" datatype="html">
        <source>
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">3</context>
        </context-group>
      </trans-unit>
      <trans-unit id="1bd7be430ac3f26a91a30d76868397fa8fea9882" datatype="html">
        <source>Hier einige Links, die den Anfang erleichtern: </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">8</context>
        </context-group>
      </trans-unit>
...
```

Following the angular i18n tutorial the next steps would be to

- create a copy of this file for every language you want to support.
- translate it.

This is where `xliffmerge` comes into play. If you have a look at the generated script, you see that after the normal call of `ng xi18n` there is a following call `ng run sampleapp:xliffmerge`.

`ng run` is an angular-cli command to run a specific builder (called architect target in cli terminology). This builder is added to the workspace definition file `angular.json`. In the section `architect` in this file you will find something like this:

```
        "xliffmerge": {
          "builder": "@ngx-i18nsupport/tooling:xliffmerge",
          "options": {
            "xliffmergeOptions": {
              "i18nFormat": "xlf",
              "srcDir": "src/i18n",
              "genDir": "src/i18n",
              "defaultLanguage": "de",
              "languages": [
                "de",
                "en"
              ]
            }
          }
        }
```

This is the builder definition. In the options section you find some configuration options and the languages used. What options are available is discussed in detail in the usage section.

Generating language specific files

After running `npm run extract-i18n` you will see some warnings:

```
WARNING: please translate file "src/i18n/messages.en.xlf" to target-language="en
```

In addition to the `ng xi18n` command the xliffmerge architect target has generated a file for your target language "en". The warning tells you that you have to translate the English file.

You will now find 3 files under src/i18n

1. `messages.xlf`: The master file containing all the messages found in your app.
2. `messages.de.xlf`: The German version.
3. `messages.en.xlf`: The (up to now untranslated) English version.

If you open the files with a text editor, you will see

- `messages.xlf` is the file Angular created for you. There are no `<target>` elements (no translation), the attribute `source-language="de"` tells the source language.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="de" datatype="plaintext" original="ng2.template">
    <body>
...
      <trans-unit id="1bd7be430ac3f26a91a30d76868397fa8fea9882" datatype="html">
        <source>Hier einige Links, die den Anfang erleichtern: </source>
...
```

- `messages.de.xlf` contains a copy, but there are `<target>` elements that contain the same values as the source elements. There is an additional attribute `state="final"`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="de" datatype="plaintext" original="ng2.template" target-
language="de">
    <body>
      <trans-unit id="91fcc40bc784cacec7c70d5f1eee77ebc1d8d308" datatype="html">
        <source>
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">3</context>
        </context-group>
      <target state="final">
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </target></trans-unit>
      <trans-unit id="1bd7be430ac3f26a91a30d76868397fa8fea9882" datatype="html">
        <source>Hier einige Links, die den Anfang erleichtern: </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">8</context>
        </context-group>
      <target state="final">Hier einige Links, die den Anfang erleichtern:
</target></trans-unit>
    </body>
  </file>
</xliff>
```

- `messages.en.xlf` is nearly the same, but the `target-language` is set to `en` and all `<target>` elements have an attribute `state="new"`. This marks the fact that you have to translate it by your own.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
```

```
  <file source-language="de" datatype="plaintext" original="ng2.template" target-
language="en">
    <body>
      <trans-unit id="91fcc40bc784cacec7c70d5f1eee77ebc1d8d308" datatype="html">
        <source>
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">3</context>
        </context-group>
      <target state="new">
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </target></trans-unit>
      <trans-unit id="1bd7be430ac3f26a91a30d76868397fa8fea9882" datatype="html">
        <source>Hier einige Links, die den Anfang erleichtern: </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">8</context>
        </context-group>
      <target state="new">Hier einige Links, die den Anfang erleichtern: </target>
</trans-unit>
    </body>
  </file>
</xliff>
```

Translate your messages files

Next you have to translate `messages.en.xlf` by yourself. Use a translation tool, if you want. E.g. have a look at Tiny Translator. At the end you just replace the target elements with the translated version and you change the state attribute to `state="translated"` or even better to `state="final"` to mark it as fully reviewed.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="de" datatype="plaintext" original="ng2.template" target-
language="en">
    <body>
      <trans-unit id="91fcc40bc784cacec7c70d5f1eee77ebc1d8d308" datatype="html">
        <source>
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">3</context>
        </context-group>
      <target state="final">
```

```
    Welcome to <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </target></trans-unit>
      <trans-unit id="1bd7be430ac3f26a91a30d76868397fa8fea9882" datatype="html">
        <source>Hier einige Links, die den Anfang erleichtern: </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">8</context>
        </context-group>
      <target state="final">Here are some links to start with: </target></trans-
unit>
    </body>
  </file>
</xliff>
```

Do not forget to put the translated files under configuration management, they should all be considered as source files.

Run your English version of the app

There already is a start script added to your `package.json`

```
{
  [...]
  "scripts": {
    [...]
    "extract-i18n": ...,
    "start-en": "ng serve --configuration=en"
  }
  [...]
}
```

and there also is a configuration change in your `angular.json`

```
.."build": {
    "configurations": {
..
            "en": {
              "aot": true,
              "outputPath": "dist/sampleapp-en",
              "i18nFile": "src/i18n/messages.en.xlf",
              "i18nFormat": "xlf",
              "i18nLocale": "en"
            }
.."serve": {
..
          "configurations": {
```

```
..
            "en": {
              "browserTarget": "sampleapp:build:en"
            }
```

Run

`npm run start-en`

You now will see the English version of your app:

Welcome to app

Here are some links to start with

Congratulations!

Change your app and do it again

In the next steps you want to add new features to your app and that includes adding some new messages.

For example change `src/app/app.component.html` to

```
...
<div style="text-align:center">
  <h1 i18n>
    Willkommen zu {{ title }}!
  </h1>
  <img ..>
</div>
<h2 i18n>Hier einige Links, die den Anfang erleichtern: </h2>
<p i18n="Beschreibung der Anwendung">Diese Anwendung ist eine reine Demonstration und
...
```

We just added another paragraph. And we used a description value for the i18n attribute.

Now extract the messages.xlf once again:

```
npm run extract-i18n
```

There will be some warnings again:

```
WARNING: merged 1 trans-units from master to "de"
WARNING: merged 1 trans-units from master to "en"
WARNING: please translate file "src/i18n/messages.en.xlf" to target-language="en
```

The warnings show you that there are new messages merged in (only 1 here).

Have a look at the newly generated xlf files (if they are under version control, you can directly look at the changes done), especially the English one:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
  <file source-language="de" datatype="plaintext" original="ng2.template" target-langu
    <body>
   <trans-unit id="91fcc40bc784cacec7c70d5f1eee77ebc1d8d308" datatype="html">
        <source>
    Willkommen zu <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">3</context>
        </context-group>
      <target state="translated">
    Welcome to <x id="INTERPOLATION" equiv-text="{{ title }}"/>!
  </target></trans-unit>
      <trans-unit id="1bd7be430ac3f26a91a30d76868397fa8fea9882" datatype="html">
        <source>Hier einige Links, die den Anfang erleichtern: </source>
        <context-group purpose="location">
          <context context-type="sourcefile">app/app.component.ts</context>
          <context context-type="linenumber">8</context>
        </context-group>
      <target state="translated">Here are some links to start with: </target></trans-u
    <trans-unit id="fb74bbbdf6190dc80f48622910221fd07ae0c70d" datatype="html">
        <source>Diese Anwendung ist eine reine Demonstration und hat keine wirklich nu
        <target state="new">Diese Anwendung ist eine reine Demonstration und hat keine
        <note priority="1" from="description">Beschreibung der Anwendung</note>
      </trans-unit></body>
  </file>
</xliff>
```

You see

- the old parts are just there. The already done parts of the translation are **untouched**.
- the new parts contain the German original as translation value and are marked with `state="new"`
- the description attribute is contained as a `<note>` element. Translation tools will show it to support the translator.

In the next step you can send the file to your translator to do the work with the new parts (or you can do that by your own). If this should take some time, you can just run the application with the partly translated file, which is better than no version at all.

This is the incompletely translated versions output:

```
# Welcome to app


## Here are some links to start with

Diese Anwendung ist eine reine Demonstration und hat keine wirklich nutzbare Funktion.
```

After the translation is done, just run the new version.

`npm run start-en`

OK, you got it:

```
# Welcome to app
```

Here are some links to start with

This application is just for demonstration purposes. It really has no value.

▸ Pages 8

Clone this wiki locally

https://github.com/martinroob/ngx-i18nsupport.wiki.git