Contents

- Angular Compiler
- Angular Translate
- I18N Pipes

Angular Compiler

Angular Compiler

- Extracts text from templates into xml files
- You can translate xml files (with a tool)
  - XML Message Bundle (XMB)
  - XML Localisation Interchange File Format (XLIFF, version 1.2)
- Compiles XML back into the application
  - JIT or AOT
- Very high performance
- You cannot change language without reloading the application
  - JIT: Reload application and use other language file
  - AOT: One build for each language

Flag texts

```
<th i18n>From</th>
<th i18n="Description">To</th>
<th i18n="Meaning|Description">Passengers</th>
```


Extract texts

```
ng-xi18n -p tsconfig.translate.json
```

Translations and AOT

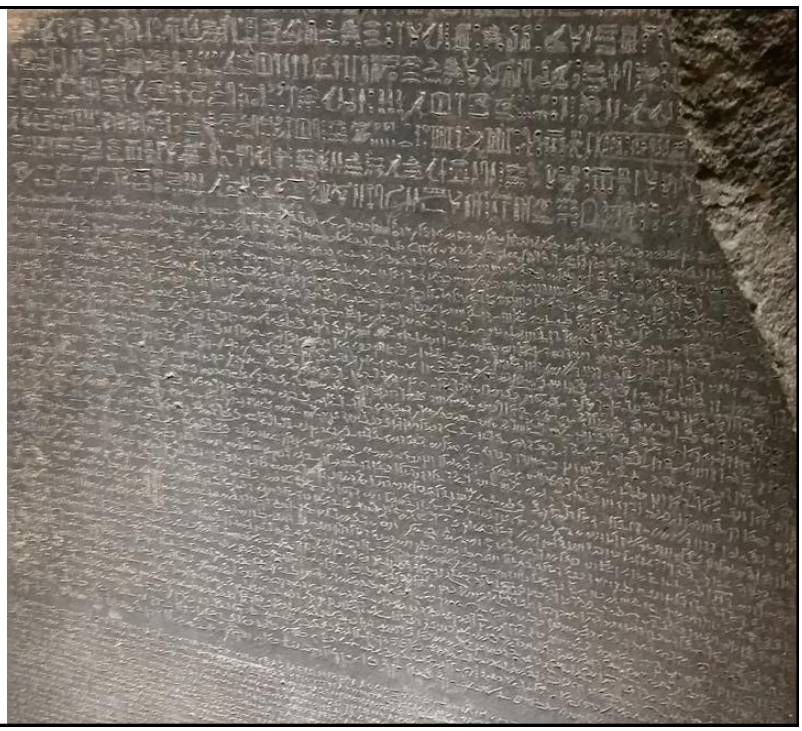
```
ngc -p tsconfig.aot.json --i18nFile=src/i18n/messages.de.xlf
                         --locale=de
                         --i18nFormat=xlf
```


Translations and JIT

```
providers: [
    {provide: TRANSLATIONS, useValue: require('./i18n/messages.de.xlf')},
    {provide: TRANSLATIONS_FORMAT, useValue: 'xlf'},
    {provide: LOCALE_ID, useValue: 'de'}
]
```

DEMO

Angular 2
Translate



Angular 2 Translate

- Community Solution
- Light weight and extensible
- https://github.com/ocombe/ng2-translate
- Migration of a very popular translation lib for AngularJS 1.x
- JSON-based Files
  - Extension points and extensions for other formats
- Changing language on the fly
- Uses data binding (performance?)
- Community Extension to extract texts from templates

Import Library

```
export function createLoader(http: Http) {
   return new TranslateStaticLoader(http, './i18n', '.json');
}
```

Import Library

```
export function createLoader(http: Http) {
   return new TranslateStaticLoader(http, './i18n', '.json');
}

@NgModule({
   imports: [
      TranslateModule.forRoot({
         provide: TranslateLoader,
         useFactory: createLoader,
         deps: [Http]
      })
   ],
   [...]
})
export class AppModule {
}
```


Initialize

```
this.translate.addLangs(['en', 'de']);
this.translate.setDefaultLang('de');
this.translate.use('de');
```

Resource Files

```
{
  "FLIGHTS": {
    "from": "From",
    "to": "to",
    "search": "Search",
    "found": "{{count}} flights found."
  }
}
```


Translation

```
<th>{{'FLIGHTS.from' | translate}}</th>

<th>{{'FLIGHTS.found' | translate:{count: 7} }}</th>
```

Load Locale Data

```
import localeDe from '@angular/common/locales/de';
import localeDeAt from '@angular/common/locales/de-AT';
import localeEs from '@angular/common/locales/es';
```

Register Locale Data

```
import localeDe from '@angular/common/locales/de';
import localeDeAt from '@angular/common/locales/de-AT';
import localeEs from '@angular/common/locales/es';

import { registerLocaleData } from '@angular/common';

registerLocaleData(localeDe); // de-DE
registerLocaleData(localeDeAt); // de-AT
registerLocaleData(localeEs); // es-ES
```


Define Default Language (JIT)

```
@NgModule({
    […],
    providers: [
         { provide: LOCALE_ID, useValue: 'de' }
    ],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Define Default Language (AOT)

```
ngc -p tsconfig.aot.json --locale=de
```


Use Pipes

```
<p>Date: {{item.date | date:'long'}}</p>
<p>Date: {{item.date | date:'long':'':'es'}}</p>
```

DEMO

I18N and Inputs?

- Pipes are just for outputs
- Use Component Lib (like Angular Material)
- General Concept: Control Value Accessor