<https://github.com/ngx-translate/core>

# @ngx-translate/core

The internationalization (i18n) library for Angular.

Simple example using ngx-translate: <https://stackblitz.com/github/ngx-translate/example>

Get the complete changelog here: <https://github.com/ngx-translate/core/releases>

## Table of Contents

* [Installation](https://github.com/ngx-translate/core#installation)
* [Usage](https://github.com/ngx-translate/core#usage)
  + [Import the TranslateModule](https://github.com/ngx-translate/core#1-import-the-translatemodule)
    - [SharedModule](https://github.com/ngx-translate/core#sharedmodule)
    - [Lazy loaded modules](https://github.com/ngx-translate/core#lazy-loaded-modules)
    - [Configuration](https://github.com/ngx-translate/core#configuration)
    - [AoT](https://github.com/ngx-translate/core#aot)
  + [Define the default language for the application](https://github.com/ngx-translate/core#2-define-the-default-language-for-the-application)
  + [Init the TranslateService for your application](https://github.com/ngx-translate/core#3-init-the-translateservice-for-your-application)
  + [Define the translations](https://github.com/ngx-translate/core#4-define-the-translations)
  + [Use the service, the pipe or the directive](https://github.com/ngx-translate/core#5-use-the-service-the-pipe-or-the-directive)
  + [Use HTML tags](https://github.com/ngx-translate/core#6-use-html-tags)
* [API](https://github.com/ngx-translate/core#api)
  + [TranslateService](https://github.com/ngx-translate/core#translateservice)
    - [Properties](https://github.com/ngx-translate/core#properties)
    - [Methods](https://github.com/ngx-translate/core#methods)
    - [Write & use your own loader](https://github.com/ngx-translate/core#write--use-your-own-loader)
      * [Example](https://github.com/ngx-translate/core#example)
    - [How to use a compiler to preprocess translation values](https://github.com/ngx-translate/core#how-to-use-a-compiler-to-preprocess-translation-values)
    - [How to handle missing translations](https://github.com/ngx-translate/core#how-to-handle-missing-translations)
      * [Example](https://github.com/ngx-translate/core#example-1)
  + [Parser](https://github.com/ngx-translate/core#parser)
    - [Methods](https://github.com/ngx-translate/core#methods)
* [FAQ](https://github.com/ngx-translate/core#faq)
  + [I'm getting an error npm ERR! peerinvalid Peer [...]](https://github.com/ngx-translate/core#im-getting-an-error-npm-err-peerinvalid-peer-)
* [Plugins](https://github.com/ngx-translate/core#plugins)
* [Editors](https://github.com/ngx-translate/core#editors)
* [Additional Framework Support](https://github.com/ngx-translate/core#additional-framework-support)

## Installation

First you need to install the npm module:

npm install @ngx-translate/core --save

Choose the version corresponding to your Angular version:

| **Angular** | **@ngx-translate/core** | **@ngx-translate/http-loader** |
| --- | --- | --- |
| 10 | 13.x+ | 6.x+ |
| 9 | 12.x+ | 5.x+ |
| 8 | 12.x+ | 4.x+ |
| 7 | 11.x+ | 4.x+ |
| 6 | 10.x | 3.x |
| 5 | 8.x to 9.x | 1.x to 2.x |
| 4.3 | 7.x or less | 1.x to 2.x |
| 2 to 4.2.x | 7.x or less | 0.x |

**If you use SystemJS** to load your files, you can check the [plunkr example](https://plnkr.co/edit/XXwyUYS6ZL7qVD9I2l0g?p=preview) for a working setup that uses the cdn <https://unpkg.com/>. If you're importing directly from node\_modules, you should edit your systemjs config file and add '@ngx-translate/core': 'node\_modules/@ngx-translate/core/bundles' in the map and '@ngx-translate/core' : { defaultExtension: 'js' } in packages.

## Usage

#### 1. Import the TranslateModule:

Finally, you can use ngx-translate in your Angular project. You have to import TranslateModule.forRoot() in the root NgModule of your application.

The [forRoot](https://angular.io/api/router/RouterModule" \l "forroot) static method is a convention that provides and configures services at the same time. Make sure you only call this method in the root module of your application, most of the time called AppModule. This method allows you to configure the TranslateModule by specifying a loader, a parser and/or a missing translations handler.

import {BrowserModule} from '@angular/platform-browser';

import {NgModule} from '@angular/core';

import {TranslateModule} from '@ngx-translate/core';

@NgModule({

imports: [

BrowserModule,

TranslateModule.forRoot()

],

bootstrap: [AppComponent]

})

export class AppModule { }

##### SharedModule

If you use a [SharedModule](https://angular.io/guide/sharing-ngmodules) that you import in multiple other feature modules, you can export the TranslateModule to make sure you don't have to import it in every module.

@NgModule({

exports: [

CommonModule,

TranslateModule

]

})

export class SharedModule { }

Note: Never call a forRoot static method in the SharedModule. You might end up with different instances of the service in your injector tree. But you can use forChild if necessary.

##### Lazy loaded modules

When you lazy load a module, you should use the forChild static method to import the TranslateModule.

Since lazy loaded modules use a different injector from the rest of your application, you can configure them separately with a different loader/compiler/parser/missing translations handler.

To make a child module extend translations from parent modules use extend: true. This will cause the service to also use translations from its parent module.

You can also isolate the service by using isolate: true. In which case the service is a completely isolated instance (for translations, current lang, events, ...). Otherwise, by default, it will share its data with other instances of the service (but you can still use a different loader/compiler/parser/handler even if you don't isolate the service).

@NgModule({

imports: [

TranslateModule.forChild({

loader: {provide: TranslateLoader, useClass: CustomLoader},

compiler: {provide: TranslateCompiler, useClass: CustomCompiler},

parser: {provide: TranslateParser, useClass: CustomParser},

missingTranslationHandler: {provide: MissingTranslationHandler, useClass: CustomHandler},

isolate: true

})

]

})

export class LazyLoadedModule { }

##### Configuration

By default, there is no loader available. You can add translations manually using setTranslation but it is better to use a loader. You can write your own loader, or import an existing one. For example you can use the [TranslateHttpLoader](https://github.com/ngx-translate/http-loader) that will load translations from files using HttpClient.

To use it, you need to install the http-loader package from @ngx-translate:

npm install @ngx-translate/http-loader --save

**NB: if you're still on Angular <4.3, please use Http from @angular/http with http-loader@0.1.0.**

Once you've decided which loader to use, you have to setup the TranslateModule to use it.

Here is how you would use the TranslateHttpLoader to load translations from "/assets/i18n/[lang].json" ([lang] is the lang that you're using, for english it could be en):

import {NgModule} from '@angular/core';

import {BrowserModule} from '@angular/platform-browser';

import {HttpClientModule, HttpClient} from '@angular/common/http';

import {TranslateModule, TranslateLoader} from '@ngx-translate/core';

import {TranslateHttpLoader} from '@ngx-translate/http-loader';

import {AppComponent} from './app';

// AoT requires an exported function for factories

export function HttpLoaderFactory(http: HttpClient) {

return new TranslateHttpLoader(http);

}

@NgModule({

imports: [

BrowserModule,

HttpClientModule,

TranslateModule.forRoot({

loader: {

provide: TranslateLoader,

useFactory: HttpLoaderFactory,

deps: [HttpClient]

}

})

],

bootstrap: [AppComponent]

})

export class AppModule { }

##### AoT

If you want to configure a custom TranslateLoader while using [AoT compilation](https://angular.io/docs/ts/latest/cookbook/aot-compiler.html) or [Ionic](http://ionic.io/), you must use an exported function instead of an inline function.

export function createTranslateLoader(http: HttpClient) {

return new TranslateHttpLoader(http, './assets/i18n/', '.json');

}

@NgModule({

imports: [

BrowserModule,

HttpClientModule,

TranslateModule.forRoot({

loader: {

provide: TranslateLoader,

useFactory: (createTranslateLoader),

deps: [HttpClient]

}

})

],

bootstrap: [AppComponent]

})

export class AppModule { }

#### 2. Define the default language for the application

@NgModule({

imports: [

BrowserModule,

TranslateModule.forRoot({

defaultLanguage: 'en'

})

],

providers: [

],

bootstrap: [AppComponent]

})

export class AppModule { }

#### 3. Init the TranslateService for your application:

import {Component} from '@angular/core';

import {TranslateService} from '@ngx-translate/core';

@Component({

selector: 'app',

template: `

<div>{{ 'HELLO' | translate:param }}</div>

`

})

export class AppComponent {

param = {value: 'world'};

constructor(translate: TranslateService) {

// this language will be used as a fallback when a translation isn't found in the current language

translate.setDefaultLang('en');

// the lang to use, if the lang isn't available, it will use the current loader to get them

translate.use('en');

}

}

#### 4. Define the translations:

Once you've imported the TranslateModule, you can put your translations in a json file that will be imported with the TranslateHttpLoader. The following translations should be stored in en.json.

{

"HELLO": "hello {{value}}"

}

You can also define your translations manually with setTranslation.

translate.setTranslation('en', {

HELLO: 'hello {{value}}'

});

The TranslateParser understands nested JSON objects. This means that you can have a translation that looks like this:

{

"HOME": {

"HELLO": "hello {{value}}"

}

}

You can then access the value by using the dot notation, in this case HOME.HELLO.

#### 5. Use the service, the pipe or the directive:

You can either use the TranslateService, the TranslatePipe or the TranslateDirective to get your translation values.

With the **service**, it looks like this:

translate.get('HELLO', {value: 'world'}).subscribe((res: string) => {

console.log(res);

//=> 'hello world'

});

This is how you do it with the **pipe**:

<div>{{ 'HELLO' | translate:param }}</div>

And in your component define param like this:

param = {value: 'world'};

You can construct the translation keys dynamically by using simple string concatenation inside the template:

<ul \*ngFor="let language of languages">

<li>{{ 'LANGUAGES.' + language | translate }}</li>

</ul>

Where languages is an array member of your component:

languages = ['EN', 'FR', 'BG'];

You can also use the output of the built-in pipes uppercase and lowercase in order to guarantee that your dynamically generated translation keys are either all uppercase or all lowercase. For example:

<p>{{ 'ROLES.' + role | uppercase | translate }}</p>

role = 'admin';

will match the following translation:

{

"ROLES": {

"ADMIN": "Administrator"

}

}

This is how you use the **directive**:

<div [translate]="'HELLO'" [translateParams]="{value: 'world'}"></div>

Or even simpler using the content of your element as a key:

<div translate [translateParams]="{value: 'world'}">HELLO</div>

#### 6. Use HTML tags:

You can easily use raw HTML tags within your translations.

{

"HELLO": "Welcome to my Angular application!<br><strong>This is an amazing app which uses the latest technologies!</strong>"

}

To render them, simply use the innerHTML attribute with the pipe on any element.

<div [innerHTML]="'HELLO' | translate"></div>

## API

### TranslateService

#### Properties:

* currentLang: The lang currently used
* currentLoader: An instance of the loader currently used (static loader by default)
* onLangChange: An EventEmitter to listen to lang change events. A LangChangeEvent is an object with the properties lang: string & translations: any (an object containing your translations).

example:

onLangChange.subscribe((event: LangChangeEvent) => {

// do something

});

* onTranslationChange: An EventEmitter to listen to translation change events. A TranslationChangeEvent is an object with the properties lang: string & translations: any (an object containing your translations).

example:

onTranslationChange.subscribe((event: TranslationChangeEvent) => {

// do something

});

* onDefaultLangChange: An EventEmitter to listen to default lang change events. A DefaultLangChangeEvent is an object with the properties lang: string & translations: any (an object containing your translations).

example:

onDefaultLangChange.subscribe((event: DefaultLangChangeEvent) => {

// do something

});

#### Methods:

* setDefaultLang(lang: string): Sets the default language to use as a fallback
* getDefaultLang(): string: Gets the default language
* use(lang: string): Observable<any>: Changes the lang currently used
* getTranslation(lang: string): Observable<any>: Gets an object of translations for a given language with the current loader
* setTranslation(lang: string, translations: Object, shouldMerge: boolean = false): Manually sets an object of translations for a given language, set shouldMerge to true if you want to append the translations instead of replacing them
* addLangs(langs: Array<string>): Add new langs to the list
* getLangs(): Returns an array of currently available langs
* get(key: string|Array<string>, interpolateParams?: Object): Observable<string|Object>: Gets the translated value of a key (or an array of keys) or the key if the value was not found
* getStreamOnTranslationChange(key: string|Array<string>, interpolateParams?: Object): Observable<string|Object>: Returns a stream of translated values of a key (or an array of keys) or the key if the value was not found. Without any onTranslationChange events this returns the same value as get but it will also emit new values whenever the translation changes.
* stream(key: string|Array<string>, interpolateParams?: Object): Observable<string|Object>: Returns a stream of translated values of a key (or an array of keys) or the key if the value was not found. Without any onLangChange events this returns the same value as get but it will also emit new values whenever the used language changes.
* instant(key: string|Array<string>, interpolateParams?: Object): string|Object: Gets the instant translated value of a key (or an array of keys). /!\ This method is **synchronous** and the default file loader is asynchronous. You are responsible for knowing when your translations have been loaded and it is safe to use this method. If you are not sure then you should use the get method instead.
* set(key: string, value: string, lang?: string): Sets the translated value of a key
* reloadLang(lang: string): Observable<string|Object>: Calls resetLang and retrieves the translations object for the current loader
* resetLang(lang: string): Removes the current translations for this lang. /!\ You will have to call use, reloadLang or getTranslation again to be able to get translations
* getBrowserLang(): string | undefined: Returns the current browser lang if available, or undefined otherwise
* getBrowserCultureLang(): string | undefined: Returns the current browser culture language name (e.g. "de-DE" if available, or undefined otherwise

#### Write & use your own loader

If you want to write your own loader, you need to create a class that implements TranslateLoader. The only required method is getTranslation that must return an Observable. If your loader is synchronous, just use [Observable.of](https://github.com/Reactive-Extensions/RxJS/blob/master/doc/api/core/operators/of.md) to create an observable from your static value.

##### Example

class CustomLoader implements TranslateLoader {

getTranslation(lang: string): Observable<any> {

return Observable.of({KEY: 'value'});

}

}

Once you've defined your loader, you can provide it in your configuration by adding it to its providers property.

@NgModule({

imports: [

BrowserModule,

TranslateModule.forRoot({

loader: {provide: TranslateLoader, useClass: CustomLoader}

})

],

bootstrap: [AppComponent]

})

export class AppModule { }

[Another custom loader example with translations stored in Firebase](https://github.com/ngx-translate/core/blob/master/FIREBASE_EXAMPLE.md)

#### How to use a compiler to preprocess translation values

By default, translation values are added "as-is". You can configure a compiler that implements TranslateCompiler to pre-process translation values when they are added (either manually or by a loader). A compiler has the following methods:

* compile(value: string, lang: string): string | Function: Compiles a string to a function or another string.
* compileTranslations(translations: any, lang: string): any: Compiles a (possibly nested) object of translation values to a structurally identical object of compiled translation values.

Using a compiler opens the door for powerful pre-processing of translation values. As long as the compiler outputs a compatible interpolation string or an interpolation function, arbitrary input syntax can be supported.

#### How to handle missing translations

You can setup a provider for the MissingTranslationHandler in the bootstrap of your application (recommended), or in the providers property of a component. It will be called when the requested translation is not available. The only required method is handle where you can do whatever you want. If this method returns a value or an observable (that should return a string), then this will be used. Just don't forget that it will be called synchronously from the instant method.

You can use useDefaultLang to decide whether default language string should be used when there is a missing translation in current language. Default value is true. If you set it to false, MissingTranslationHandler will be used instead of the default language string.

##### Example:

Create a Missing Translation Handler

import {MissingTranslationHandler, MissingTranslationHandlerParams} from '@ngx-translate/core';

export class MyMissingTranslationHandler implements MissingTranslationHandler {

handle(params: MissingTranslationHandlerParams) {

return 'some value';

}

}

Setup the Missing Translation Handler in your module import by adding it to the forRoot (or forChild) configuration.

@NgModule({

imports: [

BrowserModule,

TranslateModule.forRoot({

missingTranslationHandler: {provide: MissingTranslationHandler, useClass: MyMissingTranslationHandler},

useDefaultLang: false

})

],

providers: [

],

bootstrap: [AppComponent]

})

export class AppModule { }

### Parser

If you need it for some reason, you can use the TranslateParser service.

#### Methods:

* interpolate(expr: string | Function, params?: any): string: Interpolates a string to replace parameters or calls the interpolation function with the parameters.

This is a {{ key }} ==> This is a value with params = { key: "value" } (params) => \This is a ${params.key}` ==> This is a value with params = { key: "value" }

* getValue(target: any, key: string): any: Gets a value from an object by composed key parser.getValue({ key1: { keyA: 'valueI' }}, 'key1.keyA') ==> 'valueI'

## FAQ

#### I'm getting an error npm ERR! peerinvalid Peer [...]

If you're using npm 2.x, upgrade to npm 3.x, because npm 2 doesn't handle peer dependencies well. With npm 2 you could only use fixed versions, but with npm 3 you can use ^ to use a newer version if available.

If you're already on npm 3, check if it's an error (npm ERR!) or a warning (npm WARN!), warning are just informative and if everything works then don't worry !

If you're using an old version of Angular and ngx-translate requires a newer version then you should consider upgrading your application to use the newer angular 2 version. There is always a reason when I upgrade the minimum dependencies of the library. Often it is because Angular had a breaking changes. If it's not an option for you, then check [the changelog](https://github.com/ngx-translate/core/blob/master/releases) to know which version is the last compatible version for you.

#### I want to hot reload the translations in my application but reloadLang does not work

If you want to reload the translations and see the update on all your components without reloading the page, you have to load the translations manually and call setTranslation function which triggers onTranslationChange.

## Plugins

* [Localize Router](https://github.com/Greentube/localize-router) by @meeroslav: An implementation of routes localization for Angular. If you need localized urls (for example /fr/page and /en/page).
* [.po files Loader](https://github.com/biesbjerg/ngx-translate-po-http-loader) by @biesbjerg: Use .po translation files with ngx-translate
* [ngx-translate-extract](https://github.com/biesbjerg/ngx-translate-extract) by @biesbjerg: Extract translatable strings from your projects
* [MessageFormat Compiler](https://github.com/lephyrus/ngx-translate-messageformat-compiler) by @lephyrus: Compiler for ngx-translate that uses messageformat.js to compile translations using ICU syntax for handling pluralization, gender, and more
* [ngx-translate-zombies](https://marketplace.visualstudio.com/items?itemName=seveseves.ngx-translate-zombies) by @seveves: A vscode extension that finds unused translation keys and shows them in a diff view (so called zombies).
* [ngx-translate-multi-http-loader](https://github.com/denniske/ngx-translate-multi-http-loader) by @denniske: Fetch multiple translation files with ngx-translate.
* [ngx-translate-cache](https://github.com/jgpacheco/ngx-translate-cache) by @jgpacheco: Simplified version of localize-router. If you are already using localize-router you don't need this extension. This extension is aimed only to facilitate language caching.
* [ngx-translate-module-loader](https://github.com/larscom/ngx-translate-module-loader) by @larscom: Fetch multiple translation files (http) with ngx-translate. Each translation file gets it's own namespace out of the box and the configuration is very flexible.
* [ngx-translate-all](https://github.com/irustm/ngx-translate-all) by @irustm: Automate translations for Angular projects.
* [ngx-translate-migrate](https://github.com/irustm/ngx-translate-migrate) by @irustm: Automate migrations from ngx-translate to Angular i18n.
* [ngx-translate-lint](https://github.com/svoboda-rabstvo/ngx-translate-lint) by @svoboda-rabstvo: Simple CLI tools for check ngx-translate keys in whole app
* [ngx-translate-cut](https://github.com/bartholomej/ngx-translate-cut) by @bartholomej: Simple and useful pipe for cutting translations ✂️

## Editors

* [BabelEdit](https://www.codeandweb.com/babeledit) — translation editor for JSON files
* [Translation Manager](https://translation-manager-86c3d.firebaseapp.com/) — Progressive web-app, translation editor for JSON files
* [Crowdl.io](https://crowdl.io/) — Free translation management and crowd-translations tool with support for JSON files

### Extensions

#### VScode

* [Generate Translation](https://marketplace.visualstudio.com/items?itemName=thiagocordeirooo.generate-translation) by [@thiagocordeirooo](https://github.com/thiagocordeirooo): A visual studio code extension for you to generate the translations without leaving the current file.
* [Lingua](https://marketplace.visualstudio.com/items?itemName=chr33z.lingua-vscode&utm_source=www.vsixhub.com) by [@chr33z](https://github.com/chr33z): A visual studio code extension to help managing translations for ngx-translate - featuring inline translation lookup and in-place translation creation and editing.

## Additional Framework Support

* [Use with NativeScript](https://github.com/NathanWalker/nativescript-ng2-translate/issues/5#issuecomment-257606661)