**Internationalization and Localization – Angular**

**Steps:**

1. Add the localize package
2. Locals
3. Format data based on locale
4. Prepare component for translation
5. Work with translation files
6. Merge translations into the application
7. Deploy multiple locales
8. Import global variants of the locale data
9. Manage marked text with custom IDs
10. **Add the localize package**

ng add @angular/localize

1. **Locals**

| **Language** | **Locale** | **Unicode locale ID** |
| --- | --- | --- |
| English | Canada | en-CA |
| English | USA | en-US |
| French | Canada | fr-CA |
| French | France | fr-FR |
| Chinese | China | zh-CN |
| Chinese | Singapore | zh-SG |

1. **Format data based on locale**

Angular provides the following built-in data transformation pipes. The data transformation pipes use the LOCALE\_ID token to format data based on rules of each locale.

**Data transformation pipe Details**

DatePipe Formats a date value.

CurrencyPipe T ransforms a number into a currency string.

DecimalPipe Transforms a number into a decimal number string.

PercentPipe Transforms a number into a percentage string.

To display the current date in the format for the current locale, use the following format for the DatePipe.

**{{ today | date }}**

Override current locale for CurrencyPipe

Add the locale parameter to the pipe to override the current value of LOCALE\_ID token.

To force the currency to use American English (en-US), use the following format for the CurrencyPipe

**{{ amount | currency : 'en-US' }}**

1. **Prepare component for translation**

<https://angular.dev/guide/i18n/prepare>

To prepare your project for translation, complete the following actions.

* Use the i18n attribute to mark text in component templates

**<element i18n="{i18n\_metadata}">{string\_to\_translate}</element>**

* Use the i18n- attribute to mark attribute text strings in component templates

The following <h1> tag displays a simple English language greeting, "Hello i18n!".

**src/app/app.component.html**

**<h1>Hello i18n!</h1>**

To mark the greeting for translation, add the i18n attribute to the <h1> tag.

**src/app/app.component.html**

**<h1 i18n>Hello i18n!</h1>**

**Using conditional statement with i18n**

The following <div> tag will display translated text as part of div and aria-label based on toggle status

**HTML**

<button type="button" (click)="toggleDisplay()">Toggle</button>

<div i18n [attr.aria-label]="toggleAriaLabel()">{{toggle()}}</div>

**TS**

toggle = signal(false);

toggleAriaLabel = computed(() => {

return this.toggle()

? $localize`:Toggle Button|A button to toggle status:Show`

: $localize`:Toggle Button|A button to toggle status:Hide`;

});

inc(i: number) {

...

toggleDisplay() {

this.toggle.update((toggle) => !toggle);

}

}

* Use the $localize tagged message string to mark text strings in component code

1. **Work with translation files**

After you prepare a component for translation, use the [extract-i18n](https://angular.dev/cli/extract-i18n) [Angular CLI](https://angular.dev/cli) command to extract the marked text in the component into a *source language* file.

The marked text includes text marked with i18n, attributes marked with i18n-*attribute*, and text tagged with $localize as described in [Prepare component for translation](https://angular.dev/guide/i18n/prepare).

Complete the following steps to create and update translation files for your project.

1. [Extract the source language file](https://angular.dev/guide/i18n/translation-files#extract-the-source-language-file).
   1. Optionally, change the location, format, and name.
2. Copy the source language file to [create a translation file for each language](https://angular.dev/guide/i18n/translation-files#create-a-translation-file-for-each-language).
3. [Translate each translation file](https://angular.dev/guide/i18n/translation-files#translate-each-translation-file).
4. Translate plurals and alternate expressions separately.
   1. [Translate plurals](https://angular.dev/guide/i18n/translation-files#translate-plurals).
   2. [Translate alternate expressions](https://angular.dev/guide/i18n/translation-files#translate-alternate-expressions).
   3. [Translate nested expressions](https://angular.dev/guide/i18n/translation-files#translate-nested-expressions).

**To extract the source language file, complete the following actions.**

1. Open a terminal window.
2. Change to the root directory of your project.
3. Run the following CLI command.

adev/src/content/examples/i18n/doc-files/commands.sh

check

**ng extract-i18n**

The extract-i18n command creates a source language file named **messages**.**xlf** in the root directory of your project. For more information about the XML Localization Interchange File Format (XLIFF, version 1.2), see [XLIFF](https://en.wikipedia.org/wiki/XLIFF).

Use the following [extract-i18n](https://angular.dev/cli/extract-i18n) command options to change the source language file location, format, and file name.

1. **Merge translations into the application**
2. **Deploy multiple locales**
3. **Import global variants of the locale data**
4. **Manage marked text with custom IDs**