Prepare component for translation

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

- Use the `i18n` attribute to mark text in component templates
- Use the `i18n-` attribute to mark attribute text strings in component templates
- Use the `$localize` tagged message string to mark text strings in component code

Mark text in component template

In a component template, the i18n metadata is the value of the `i18n` attribute.

<element i18n="{i18n_metadata}">{string_to_translate}</element>

Use the `i18n` attribute to mark a static text message in your component templates for translation. Place it on every element tag that contains fixed text you want to translate.

The `i18n` attribute is a custom attribute that the Angular tools and compilers recognize.

`i18n` example

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

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

Translate inline text without HTML element

Use the `<ng-container>` element to associate a translation behavior for specific text without changing the way text is displayed.

Each HTML element creates a new DOM element. To avoid creating a new DOM element, wrap the text in an `<ng-container>` element. The following example shows the `<ng-container>` element transformed into a non-displayed HTML comment.

Mark element attributes for translations

In a component template, the i18n metadata is the value of the `i18n-{attribute_name}` attribute.

<element i18n-{attribute_name}="{i18n_metadata}" {attribute_name}="{attribute_value}" />

The attributes of HTML elements include text that should be translated along with the rest of the displayed text in the component template.

Use `i18n-{attribute_name}` with any attribute of any element and replace `{attribute_name}` with the name of the attribute. Use the following syntax to assign a meaning, description, and custom ID.

i18n-{attribute_name}="{meaning}|{description}@@{id}"

`i18n-title` example

To translate the title of an image, review this example. The following example displays an image with a `title` attribute.

To mark the title attribute for translation, complete the following action.

1. Add the `i18n-title` attribute

The following example displays how to mark the `title` attribute on the `img` tag by adding `i18n-title`.

Mark text in component code

In component code, the translation source text and the metadata are surrounded by backtick (`` ` ``) characters.

Use the `$localize` tagged message string to mark a string in your code for translation.

$localize `string_to_translate`;

The i18n metadata is surrounded by colon (`:`) characters and prepends the translation source text.

$localize `:{i18n_metadata}:string_to_translate`

Include interpolated text

Include interpolations in a `$localize` tagged message string.

$localize `string_to_translate ${variable_name}`;

Name the interpolation placeholder

$localize `string_to_translate ${variable_name}:placeholder_name:`;

i18n metadata for translation

{meaning}|{description}@@{custom_id}

The following parameters provide context and additional information to reduce confusion for your translator.

Metadata parameter	Details
Custom ID	Provide a custom identifier
Description	Provide additional information or context
Meaning	Provide the meaning or intent of the text within the specific context

For additional information about custom IDs, see Manage marked text with custom IDs.

Add helpful descriptions and meanings

To translate a text message accurately, provide additional information or context for the translator.

Add a *description* of the text message as the value of the `i18n` attribute or `$localize` tagged message string.

The following example shows the value of the `i18n` attribute.

The following example shows the value of the `$localize` tagged message string with a description.

$localize `:An introduction header for this sample:Hello i18n!`;

The translator may also need to know the meaning or intent of the text message within this particular application context, in order to translate it the same way as other text with the same meaning. Start the `i18n` attribute value

with the *meaning* and separate it from the *description* with the `|` character: `{meaning}|{description}` .

`h1` example

For example, you may want to specify that the `<h1>` tag is a site header that you need translated the same way, whether it is used as a header or referenced in another section of text.

The following example shows how to specify that the `<h1>` tag must be translated as a header or referenced elsewhere.

The result is any text marked with `site header` , as the *meaning* is translated exactly the same way.

The following code example shows the value of the `$localize` tagged message string with a meaning and a description.

$localize `:site header|An introduction header for this sample:Hello i18n!` ;

How meanings control text extraction and merges
The Angular extraction tool generates a translation unit entry for each `i18n` attribute in a template. The Angular extraction tool assigns each translation unit a unique ID based on the *meaning* and *description*.

For more information about the Angular extraction tool, see Work with translation files.

The same text elements with different *meanings* are extracted with different IDs. For example, if the word "right" uses the following two definitions in two different locations, the word is translated differently and merged back into the application as different translation entries.

- `correct` as in "you are right"
- `direction` as in "turn right"

If the same text elements meet the following conditions, the text elements are extracted only once and use the same ID.

- Same meaning or definition
- Different descriptions

That one translation entry is merged back into the application wherever the same text elements appear.

ICU expressions

ICU expressions help you mark alternate text in component templates to meet conditions. An ICU expression includes a component property, an ICU clause, and the case statements surrounded by open curly brace (`{`) and close curly brace (`}`) characters.

{ component_property, icu_clause, case_statements }

The component property defines the variable An ICU clause defines the type of conditional text.

ICU clause	Details
`plural`	Mark the use of plural numbers
`select`	Mark choices for alternate text based on your defined string values

To simplify translation, use International Components for Unicode clauses (ICU clauses) with regular expressions.

The ICU clauses adhere to the ICU Message Format specified in the CLDR pluralization rules.

Mark plurals

Different languages have different pluralization rules that increase the difficulty of translation. Because other locales express cardinality differently, you may need to set pluralization categories that do not align with English. Use the `plural` clause to mark expressions that may not be meaningful if translated word-for-word.

{ component_property, plural, pluralization_categories }

After the pluralization category, enter the default text (English) surrounded by open curly brace (`{`) and close curly brace (`}`) characters.

pluralization_category { }

The following pluralization categories are available for English and may change based on the locale.

Pluralization category	Details	Example
`zero`	Quantity is zero	`=0 { }` `zero { }`
`one`	Quantity is 1	`=1 { }` `one { }`
`two`	Quantity is 2	`=2 { }` `two { }`
`few`	Quantity is 2 or more	`few { }`
`many`	Quantity is a large number	`many { }`
`other`	The default quantity	`other { }`

If none of the pluralization categories match, Angular uses `other` to match the standard fallback for a missing category.

other { default_quantity }

For more information about pluralization categories, see Choosing plural category names in the CLDR - Unicode Common Locale Data Repository.

Background: Locales may not support some pluralization categories
Many locales don't support some of the pluralization categories. The default locale (`en-US`) uses a very simple `plural()` function that doesn't support the `few` pluralization category. Another locale with a simple `plural()` function is `es` . The following code example shows the en-US `plural()` function.

The `plural()` function only returns 1 (`one`) or 5 (`other`). The `few` category never matches.

`minutes` example

If you want to display the following phrase in English, where `x` is a number.

updated x minutes ago

And you also want to display the following phrases based on the cardinality of `x` .

updated just now

updated one minute ago

Use HTML markup and interpolations. The following code example shows how to use the `plural` clause to express the previous three situations in a `<span>` element.

Review the following details in the previous code example.

Parameter	Details
`minutes`	The first parameter specifies the component property is `minutes` and determines the number of minutes.
`plural`	The second parameter specifies the ICU clause is `plural`.
`=0 {just now}`	For zero minutes, the pluralization category is `=0`. The value is `just now`.
`=1 {one minute}`	For one minute, the pluralization category is `=1`. The value is `one minute`.
`other {{{minutes}} minutes ago}`	For any unmatched cardinality, the default pluralization category is `other`. The value is `{{minutes}} minutes ago`.

`{{minutes}}` is an interpolation.

Mark alternates and nested expressions

The `select` clause marks choices for alternate text based on your defined string values.

{ component_property, select, selection_categories }

Translate all of the alternates to display alternate text based on the value of a variable.

After the selection category, enter the text (English) surrounded by open curly brace (`{`) and close curly brace (`}`) characters.

selection_category { text }

Different locales have different grammatical constructions that increase the difficulty of translation. Use HTML markup. If none of the selection categories match, Angular uses `other` to match the standard fallback for a missing category.

other { default_value }

`gender` example

If you want to display the following phrase in English.

The author is other

And you also want to display the following phrases based on the `gender` property of the component.

The author is female

The author is male

The following code example shows how to bind the `gender` property of the component and use the `select` clause to express the previous three situations in a `<span>` element.

The `gender` property binds the outputs to each of following string values.

Value	English value
female	`female`
male	`male`
other	`other`

The `select` clause maps the values to the appropriate translations. The following code example shows `gender` property used with the select clause.

`gender` and `minutes` example

Combine different clauses together, such as the `plural` and `select` clauses. The following code example shows nested clauses based on the `gender` and `minutes` examples.

What's next

- Work with translation files

@reviewed 2021-12-13