Internationalization Guidelines

- Basic strings
- Variables
- Plurals
  - Basic Pluralization
  - Pluralization done later
- Disambiguation by context
- Descriptions
- Newline characters
- Empty strings
- Escaping strings
- Best Practices for writing strings

In this article, you will learn when and how to use all available i18n functions and the best practices for writing your strings.

The recommendations in this article applies both for your PHP and your javascript code. You can see all the available functions for each language in the I18n functions page. The functions available for javascript will also be highlighted.

Basic strings

The most commonly used function is `__()`. It returns the translation of its argument:

```
__( 'Blog Options', 'my-theme' );
```

Another simple one is `_e()`, which outputs the translation of its argument. Instead of writing:

```
echo __( 'WordPress is the best!', 'my-theme' );
```

you can use the shorter:

```
_e( 'WordPress is the best!', 'my-theme' );
```

`__()` is also available for javascript

Variables

If you are using variables in strings, similar to the example below, you need to use placeholders.

```
echo 'Your city is $city.'
```

Use the `printf` family of functions. Especially helpful are `printf` and `sprintf`. For example:

```
printf(
    /* translators: %s: Name of a city */
    __( 'Your city is %s.', 'my-theme' ),
    $city
);
```

Notice that the string for translation is the template `"Your city is %s."`, which is the same in both the source and at run-time.

If you have more than one placeholder in a string, it is recommended that you use argument swapping. In this case, single quotes `(')` are mandatory : double quotes `(")` tell php to interpret the `$s` as the `s` variable, which is not what we want.

```
printf(
    /* translators: 1: Name of a city 2: ZIP code */
    __( 'Your city is %1$s, and your zip code is %2$s.', 'my-theme' ),
    $city,
    $zipcode
);
```

Here the zip code is displayed after the city name. In some languages, displaying the zip code and city in opposite order is more appropriate. Using `%s` prefix, as in the above example, allows for this. A translation can be written:

```
printf(
    /* translators: 1:ZIP code 2:Name of a city */
    __( 'Your zip code is %2$s, and your city is %1$s.', 'my-theme' ),
    $city,
    $zipcode
);
```

`sprintf` is also available for javascript translations:

```
const zipCodeMessage = sprintf(
    /* translators: 1:ZIP code 2:Name of a city */
    __( 'Your zip code is %2$s, and your city is %1$s.', 'my-theme'),
    city,
    zipcode
);
```

The following example tells you what not to do\

This is incorrect.

```
// This is incorrect do not use.
_e( "Your city is $city.", 'my-theme' );
```

The strings for translation are extracted from the source without executing the PHP associated with it. For example: The variable `$city` may be Vancouver, so your string will read `"Your city is Vancouver"` when the template is run but gettext won't know what is inside the PHP variable in advance.

As the value of the variable is unknown when your string is translated, it would require the translator to know every case for the variable `$country`. This is not ideal, and it is best to remove dynamic content and allow translators to focus on static content.

Plurals

Basic Pluralization

If you have a string that changes when the number of items changes. In English you have `"One comment"` and `"Two comments"`. In other languages, you can have multiple plural forms. To handle this in WordPress, you can use the `_n()` function.

```
printf(
    _n(
        '%s comment',
        '%s comments',
        get_comments_number(),
        'my-theme'
    ),
    number_format_i18n( get_comments_number() )
);
```

`_n()` accepts 4 arguments:

- singular – the singular form of the string (note that it can be used for numbers other than one in some languages, so `'%s item'` should be used instead of `'One item'`)
- plural – the plural form of the string
- count – the number of objects, which will determine whether the singular or the plural form should be returned (there are languages, which have far more than 2 forms)
- text domain – the theme's text domain

The return value of the functions is the correct translated form, corresponding to the given count.

`_n() is also available for javascript

Pluralization done later

You first set the plural strings with `_n_noop()` or `_nx_noop()`.

```
$comments_plural = _n_noop(
    '%s comment.',
    '%s comments.'
);
```

At a later point in the code, you can use `translate_nooped_plural()` to load the strings.

```
printf(
    translate_nooped_plural(
        $comments_plural,
        get_comments_number(),
        'my-theme'
```

```
    ),
    number_format_i18n( get_comments_number() )
);
```

Disambiguation by context

Sometimes a term is used in more than one context and must be translated separately in other languages, even though the same word is used for each context in English. For example, the word `Post` can be used both as a verb `"Click here to post your comment"` and as a noun `"Edit this Post"`. In such cases the `_x()` or `_ex()` function should be used. It is similar to `__()` and `_e()`, but it has an additional argument — the context:

```
_x( 'Post', 'noun', 'my-theme' );
_x( 'post', 'verb', 'my-theme' );
```

Using this method in both cases, we get the string Comment for the original version. However, translators will see two Comment strings for translation, each in a different context.

Taking an example from the German version of WordPress as an illustration: Post is Beiträge. The corresponding verb form in German is beitragen.

Note that similar to `__()`, `_x()` has an `echo` version: `_ex()`. The previous example could be written as:

```
_ex( 'Post', 'noun', 'my-theme' );
_ex( 'post', 'verb', 'my-theme' );
```

Use the one you feel enhances legibility and ease of coding.

`_x() and _nx()` are also available for javascript

Descriptions

You can add a clarifying comment in the source code, so translators know how to translate a string like `__( 'g:i:s a' )` . It must start with the word `translators:`, in all lowercase, and be the last PHP comment before the gettext call. Here is an example:

```
/* translators: draft saved date format, see http://php.net/date */
$saved_date_format = __( 'g:i:s a' );
```

Multi-line example:

```
/*
 * translators: Replace with a city related to your locale.
 * Test that it matches the expected location and has upcoming
 * events before including it. If no cities related to your
 * locale have events, then use a city related to your locale
 * that would be recognizable to most users.
 */
?>
<input placeholder="<?php esc_attr_e( 'Cincinnati' ); ?>" id="location" type="text"
name="location" />
```

Newline characters

Gettext doesn’t like `r` (ASCII code: 13) in translatable strings, so avoid it and use `n` instead.

Empty strings

The empty string is reserved for internal Gettext usage, and you must not try to internationalize the empty string. It also doesn’t make any sense because translators won’t have context.

If you have a valid use-case to internationalize an empty string, add context to both help translators and be in peace with the Gettext system.

Escaping strings

It is good to escape all of your strings, preventing translators from running malicious code. There are a few escape functions that are integrated with internationalization functions.

```
<a title="<?php esc_attr_e( 'Skip to content', 'my-theme' ); ?>" class="screen-reader-text
skip-link" href="#content" >
  <?php _e( 'Skip to content', 'my-theme' ); ?>
</a>
```

```
<label for="nav-menu">
  <?php esc_html_e( 'Select Menu:', 'my-theme' ); ?>
</label>
```

Best Practices for writing strings

Here are the best practices for writing strings

- Use decent English style – minimize slang and abbreviations.
- Use entire sentences – in most languages, word order is different than English.
- Split at paragraphs – merge related sentences, but do not include a whole page of text in one string.
- Do not leave leading or trailing whitespace in a translatable phrase.
- Assume strings can double in length when translated.
- Avoid unusual markup and unusual control characters – do not include tags that surround your text.
- Do not put unnecessary HTML markup into the translated string.
- Do not leave URLs for translation, unless they could have a version in another language.
- Add the variables as placeholders to the string as in some languages the placeholders change position.

```
printf(
    __( 'Search results for: %s', 'my-theme' ),
    get_search_query()
);
```

- Use format strings instead of string concatenation – translate phrases and not words –

```
printf(
    __( 'Your city is %1$s, and your zip code is %2$s.', 'my-theme' ),
    $city,
    $zipcode
);
```

is always better than

```
__( 'Your city is ', 'my-theme' ) . $city . __( ', and your zip code is ', 'my-theme' ) .
$zipcode;
```

- Try to use the same words and symbols to prevent translating multiple similar strings (e.g. don’t do the following)

```
__( 'Posts:', 'my-theme' ); and __( 'Posts', 'my-theme' );
```

□□□ #2
□ □□□ □□□ 2026-10-31 18:07:35 UTC
□ □□□ □□□ 2026-10-31 18:07:36 UTC