Internationalization

- Why internationalization is important
- How to internationalize your theme
  - Getting your text domain
  - Defining your text domain
  - Wrapping text with internationalization functions
  - Loading translations
- Resources

Internationalization is the process of developing your theme in such a way that its custom text can be translated into other languages. The term is often abbreviated as “i18n” because there are 18 letters between the “i” and “n,” the first and last letters.

In this article, you will learn how to internationalize your WordPress themes and why it is important.

Why internationalization is important

WordPress is used all over the world, in countries where your native language is not used. The strings in the WordPress themes need to be coded in a special way so that they can be easily translated into other languages.

Even if you yourself do not know multiple languages, internationalization leaves the door open for translators to localize the text. Localization is the process of taking internationalized text and translating it for a specific language and locale. It is often abbreviated as “l10n.”

WordPress has a built-in translation system that allows localizing a theme without having to modify the source code of the theme itself. This is important because it means that users can keep up with theme updates while using their preferred translation.

You can certainly build a WordPress theme without internationalization, particularly if it’s for a site that you know will never be translated. But in most cases, it is considered standard practice to

internationalize all of your theme's text. For one, it means that it can be used by people in other languages, giving you a wider audience. And it is also a requirement when submitting a theme to the official Theme Directory.

How to internationalize your theme

Getting your text domain

The text domain is a unique identifier, allowing WordPress to distinguish between all of the loaded translations. That means that you need something unique for your theme so that WordPress will recognize translations that belong to it.

The text domain is used in three different places in your theme:

- In the `style.css` file header.
- As an argument in internationalization functions.
- As an argument when loading translations (optional for themes submitted to the WordPress Theme Directory).

Your theme's text domain should always match your theme slug, as described in the Reading this handbook documentation. This is a central component in making sure your theme is ready for translation.

The *de facto* standard (and requirement if submitting your theme to the WordPress Theme Directory) is to use the kebab-case version of your theme name. This means that it should:

- Use all lowercase letters.
- Have no spaces.
- Hyphenate when there are multiple words.

So if your theme's name is `Fabled Sunset`, your text domain should be `fabled-sunset`.

Defining your text domain

As described in the Main Stylesheet documentation, you can configure several important pieces of metadata about your theme via the `style.css` file header. In particular, there are two meta values you can define related to internationalization:

- **Text Domain:** The string used for the text domain for translations.

- **Domain Path:** A relative path to where theme translations are stored. WordPress uses this field when the theme is disabled to detect translations. Defaults to `/languages` if not defined.

For translations to work correctly, you must at least define the `Text Domain` field, which should be your theme's slug.

Let's imagine your theme name is "Fabled Sunset" and you want translations stored in your theme's `/assets/lang` folder. Your theme's `style.css` should define these fields like so:

```
/**
 * Theme Name:        Fabled Sunset
 * ...
 * Text Domain:       fabled-sunset
 * Domain Path:       /assets/lang
 * ...
 */
```

It's important to add this metadata to your `style.css` file because WordPress uses it even when the theme is not enabled (for example, when viewing all themes from the **Appearance > Themes** screen in the admin).

Wrapping text with internationalization functions

For text in your theme to be easily translated in your theme, it must be wrapped in an internationalization function instead of hardcoded. The available internationalization functions are listed in the Common APIs handbook.

In a typical HTML-based webpage, you would add plain text strings. For example, take a look at this example heading:

```
<h2>Latest Posts</h2>
```

Because it is merely plain text, it cannot be translated into other languages without directly modifying the code.

Let's wrap that text in the `_e()` internationalization function, which tells WordPress that the text inside should be available for translation and echoed (printed to the screen):

```
<h2><?php _e( 'Latest Posts', 'themeslug' ); ?></h2>
```

Now users can add translations for your theme without touching the code.

As you may have noticed, the `_e()` function accepted a second parameter of `themeslug`. This should be replaced with your text domain.

There are many other internationalization functions that you will use, and you will learn about them later. For now, the idea is to introduce you to the basic concept.

Because internationalized text is technically a PHP variable, it means that it can be exploited and become a security vulnerability. You should always use the translate-and-escape functions when possible. Otherwise, be sure to wrap them in an escaping function when outputting, as described in the Security documentation.

Loading translations

For themes hosted in the WordPress Theme Directory, you do not need to load translations. WordPress automatically checks the `wp-content/languages` directory in a user's install and will download translations from the Translating WordPress site. All translations can be handled there, and you don't have to worry about storing or loading them from your theme.

Translation in WordPress are saved in `.po` (human readable) and `.mo` (machine readable) files. The `.mo` files are the ones actually used by WordPress.

WordPress will look in two places for translation files based on the user's locale defined for the site:

- `wp-content/themes/your-theme/{domain-path}/{locale}.mo`
- `wp-content/languages/themes/{textdomain}-{locale}.mo`

To load translation files from your theme, you must use one of two functions:

- `load_theme_textdomain()`
- `load_child_theme_textdomain()` (if building a child theme)

Let's take a look at the `load_theme_textdomain()` function signature (it's the same for both functions):

```
load_theme_textdomain(
string $domain,
string|false $path = false
): bool
```

The function accepts two parameters:

- `$domain`: This should be the text domain for your theme as defined in `style.css`.
- `$path`: This must be a full directory path to the location of translations in your theme. If left undefined, WordPress will fall back to your theme's root folder.

Using the same example "Fabled Sunset" theme described earlier with translations stored in the `/assets/lang` folder, let's try loading translations from the theme.

Add this code to your theme's `functions.php` file:

```
add_action( 'after_setup_theme', 'themeslug_load_textdomain' );

function themeslug_load_textdomain() {
	load_theme_textdomain(
		'fabled-sunset',
		get_parent_theme_file_path( 'assets/lang' )
	);
}
```

The above call to `load_theme_textdomain()` was executed on the `after_setup_theme` action hook, which fires immediately after the theme's `functions.php` is loaded.

If you were building a child theme, your `functions.php` code would instead look like this:

```
add_action( 'after_setup_theme', 'themeslug_load_textdomain' );

function themeslug_load_textdomain() {
	load_child_theme_textdomain(
		'fabled-sunset',
		get_theme_file_path( 'assets/lang' )
	);
}
```

Resources

Internationalization is a large topic and is covered in full in the Internationalization chapter of the Common APIs handbook. What you learn there will apply to both theme and plugin development.

WordPress has many internationalization functions for you to use when wrapping text. Which one to use will be context-specific. Take the time to study each function so that you will know when to use it. The Internationalization Guidelines will set you down the correct path.

□□□ #3
□ □□□ □□□ 2026-08-29 23:36:32 UTC
□ □□□ □□□ 2026-10-31 06:44:10 UTC