



Internationalizing WordPress Themes and Plugins

According to [GALA \(Globalization and Localization Alliance\)](#):

Internationalization is a design process that ensures a product (usually a software application) can be adapted to various languages and regions without requiring engineering changes to the source code. Think of internationalization as readiness for localization...

Some practical examples of how internationalization is critical to multilingual products include:

- Independence from a specific language/character set encoding
- Independence from specific cultural conventions
- Removal of hard-coded text
- Minimization of concatenated text strings
- Careful use of in-line variables
- Compatibility with third-party tools
- Unicode compliance for global text display
- Accommodation of double-byte languages (for example, Japanese)
- Accommodation of right-to-left languages (for example, Arabic)

In the context of WordPress, internationalization means at least two things:

- Ensuring that the text in our themes and plugins is ready for localization
- Making sure that we consider right-to-left languages

The strategies for internationalizing plugins and themes are slightly different so we'll cover them separately.

Internationalization tools in

WordPress

WordPress translation infrastructure is built on top of [GNU Gettext](#) so that's a good starting point for research. The following is WordPress Specific.

Text Domain

The text domain provides WordPress with a unique identifier for our plugin or theme. This is important because WordPress will have many plugins and a theme to sort through so having unique names makes things easier.

In a theme the text domain and the domain path, the location of our translated files, is placed on the root css style sheet. For our my-demo-theme theme this is placed in a comment inside `style.css`.

```
/*
 * Theme Name: My Theme
 * Author: Theme Author
 * Text Domain: my-demo-theme
 * Domain Path: /languages
 */
```

For plugins the Text Domain and Domain path are placed in a comment on the root PHP file, either `index.php` or the root of the plugin code.

```
/*
 * Plugin Name: My Plugin
 * Author: Plugin Author
 * Text Domain: my-demo-plugin
 * Domain Path: /languages
 */
```

i18n functions

There are several i18n functions available for PHP internationalization

The most basic one is [__\(\)](#) that will just translate its content.

```
<?php
__('Hello World', 'my-demo-theme');
```

Note that it will not echo the result, you have to use a different function (`_e()`), discussed next.

`_e()` is similar to the previous example but it also displays the output to the page.

```
<?php
_e('Hello World', 'my-demo-theme');
```

The next one, `_x()`, is similar to `__()` but it also provides a context to help translators. You could do a comment before the item being translated but this will help systems like [Polyglots](#) do a better job.

The function takes three arguments:

- The string to translate
- The context for the translation
- The text domain

```
_x( 'Read', '
    past participle: books I have read',
    'my-demo-theme'
);
```

In this example, the word `read`, on its own, has multiple meanings and the translator will not be able to get the context right away so adding the context helps with the translation.

The `_ex()` function is a combination of `_e()` and `_x()`.

```
_ex( 'Read', '
    past participle: books I have read',
    'my-demo-theme'
```

```
);
```

The next block is for strings that should be pluralized. The first one is [_n\(\)](#) and it takes a singular term, a plural term, their definition and the text domain.

```
<?php
people = sprintf(
    _n(
        '%s p' . '%s person' // Singular Form  '%s people', // '%s people' // Plural
        $count, // Number to compare to
        'my-demo-theme' // Text Domain
    ),
    number_format_i18n( $count )
    // Converts number to
    // locale appropriate version
);
```

The second parameter to `sprintf()` formats the number we produce into something appropriate to the locale the function is being called from.

[_nx\(\)](#) is a combination of `_n()` and `_x()` in that it provides a way to pluralize content and the context necessary for translators.

```
<?
$people = sprintf( _nx(
    '%s person' // Singular
    '%s people', '%s people' // Plural
    $count, // Number to compare to
    'my-demo-theme' // Text Domain', // Context
    'my-demo-theme' // Text Domain
),
    number_format_i18n( $count )
    // Converts number to
    // locale appropriate version
);
```

The final block of functions are equivalent to [_n_noop\(\)](#) and [_nx_noop\(\)](#) keep

structures with translatable plural strings and use them later when the value is known.

Once you're ready to process the noop functions, you call [translate_nooped_plural\(\)](#)

```
<?
printf(
    translate_nooped_plural(
        $people,
        $count,
        'my-demo-theme'
    ),
    number_format_i18n( $count )
);
```

Escaping HTML

Three of the functions have equivalent versions that will escape any HTML In their values, rendering the string safe to use in HTML attributes.

- [esc_html__\(\)](#)
- [esc_html_e\(\)](#)
- [esc_html_x\(\)](#)

The following code will use the translation for Hello World available in the theme represented by my-demo-theme and escape the translated string to render HTML-specific characters safe.

```
<p><?php esc_html_e(
    'Hello World!',
    'my-demo-theme'
); ?></p>
```

Variables

What if you have a string like the following:

```
<?php
echo 'Hello $city.'
```

`$city` is a variable and should not be translated as such. The solution is to use placeholders for the variable, along with the `printf` family of functions. Especially helpful are [printf](#) and [sprintf](#). Here is what one solution looks like:

```
<?php
$city = Sao Paulo;

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

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

Also note that there is a hint for translators so that they know the context of the placeholder.

Argument swapping

If you have more than one placeholder in a string, it is recommended that you use argument swapping.

With argument swapping, you must use single quotes (') around the string because double quotes (") will cause php to interpret the `$s` as the `s` variable, which is not what we want.

In the following example, the first substitution is the name of a city and the second is the zip code.

```
<?php
printf(
```

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

This will work for most western languages. However for some languages displaying the zip code and city in opposite order would be more appropriate.

Using %s prefix in the above example, allows for such a case. A translation can thereby be written:

The modified example changes the order of the variables without changing their meaning.

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

Internationalizing a plugin

After setting up the text domain in your plugin metadata we need to load the translated files to use them.

For plugins we use the [load_plugin_textdomain](#) function; it takes 4 parameters:

1. The name of text domain
2. This parameter is deprecated so always use FALSE
3. The path to the directory where your plugin's translations are stored

Then add an action for the `plugins_loaded` hook and call the function we just created.

```
<?
function rivendellweb_load_plugin_textdomain() {
```

```

load_plugin_textdomain(
    'm' . 'my-demo-plugin' // 1
    FALSE, // 2
    basename( // 3
        dirname( __FILE__ ) ) . '/languages/'
    );
}
add_action( 'plugins_loaded', 'rivendellweb_load_plugin_textdomain' );
'/languages/'

```

See [How to Internationalize Your Plugin](#) for more information about plugin internationalization.

Internationalizing a theme

Internationalizing themes is slightly different than plugins. The function we call is [load_theme_textdomain](#).

The function takes two parameters:

1. The text domain identifier for the theme we want to use
2. The location of the language file

```

<?php
function rivendellweb_load_theme_textdomain() {
    load_theme_textdomain( 'my-demo-theme', get_template_directory() . '/'
}
add_action(
    'after_setup_theme',
    'rivendellweb_load_theme_textdomain'
);

```


Internationalizing Javascript

Note

We've already covered the basics of Javascript i18n in part 3 of my [Building Gutenberg Blocks](#) series.

We'll just revisit some of the most important details here.

At this time the only use I see for JavaScript internationalization in WordPress is for Gutenberg blocks.

[wp.i18n](#) provides a subset of the Gettext localization functions discussed earlier.

- `__('Hello World', 'my-text-domain')` – Translate a string
- `_n('%s Comment', '%s Comments', numberOfComments, 'my-text-domain')` – Translate and retrieve the singular or plural form based on the supplied number.
- `_x('Default', 'block style', 'my-text-domain')` – Translate a certain string with additional context.

This example provides a basic internationalize Gutenberg block. Since we're working with React, our process has a build system that will convert the import statements into something older browsers can use.

```
import { __ } from '@wordpress/i18n';
import { registerBlockType } from '@wordpress/blocks';

registerBlockType( 'myguten/simple', {
  title: __( 'Simple Block', 'myguten' ),
  category: 'widgets',

  edit: () => {
    return (
      <p style="color:red">
        { __( 'Hello World', 'myguten' ) }
      </p>
    );
  }
});
```

```

        </p>
    );
},

save: () => {
    return (
        <p style="color:red">
            { __( 'Hello World', 'myguten' ) }
        </p>
    );
},
} );

```

This will not work with external scripts. For that we need to use [wp_localize_script](#)

The function will only work with scripts we've enqueued to the system using `wp_enqueue_script`.

```

wp_enqueue_script(
    'rivendellweb-navigation',
    get_template_directory_uri() . '/js/navigation.js',
    array('jquery'), '20151215', true
);

```

`wp_localize_script` takes three attributes:

- The name of the enqueued script
- The name of the javascript object we want to work with
- An array of internationalized objects

```

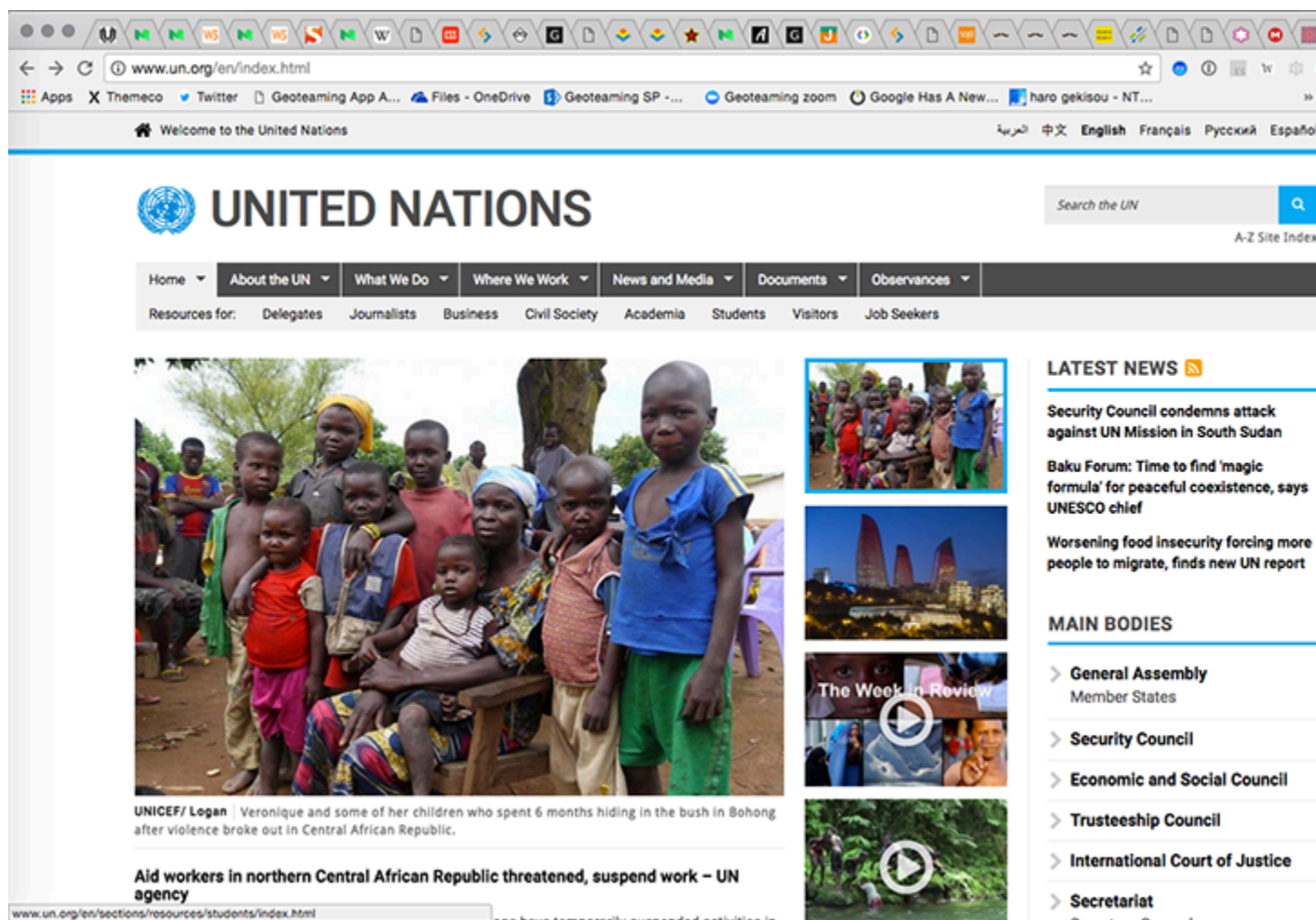
wp_localize_script( 'rivendellweb-navigation',
    'rivendellwebScreenReaderText',
    array(
        'expand' => __( 'Expand child menu', 'rivendellweb' ),
        'collapse' => __( 'Collapse child menu', 'rivendellweb' ),
    )
);

```

Additional considerations: RTL languages

If you're planning on distributing your theme and plugin there's one final consideration that I want to mention on this post: right to left language and how much our designs need to change to accommodate those languages.

Compare the next two images. The first one is in English, a left to right, top to bottom language. The second one is in Arabic, a right to left, top to bottom language.



The screenshot shows the United Nations website in English. The browser address bar displays www.un.org/en/index.html. The page features a navigation bar with links such as Home, About the UN, What We Do, Where We Work, News and Media, Documents, and Observances. A search bar is located in the top right corner. The main content area includes a large photograph of a group of children and a woman, with a caption from UNICEF/Logan stating: "Veronique and some of her children who spent 6 months hiding in the bush in Bohong after violence broke out in Central African Republic." Below this, a headline reads: "Aid workers in northern Central African Republic threatened, suspend work – UN agency". To the right, there is a "LATEST NEWS" section with headlines about the Security Council, the Baku Forum, and food insecurity. A "MAIN BODIES" section lists various UN organs. The browser's address bar also shows the URL www.un.org/en/sections/resources/students/index.html.

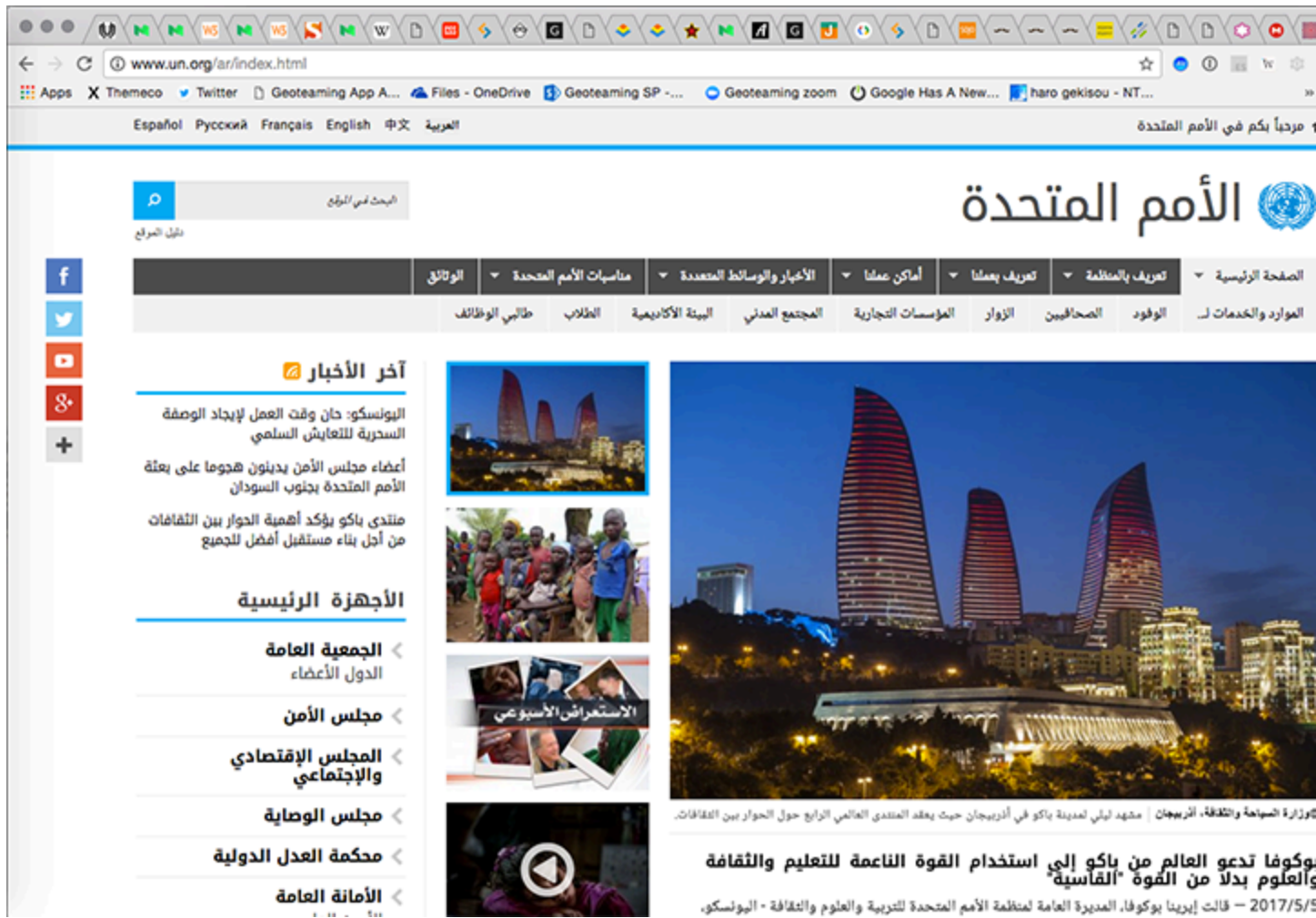


Figure 1: English and Arabic versions of the United Nations Website www.un.org

The content flows differently and we should take these languages into account when deciding on margin and padding for our content.

Because we don't know where our themes and plugins will be used we need to keep these differences in mind and provide some level of support for RTL languages.

You can convert your stylesheets to work with RTL languages manually or using tools like [gulp-rtlcss](https://github.com/jonahwilliams/gulp-rtlcss)