# Lesson information

**Goals:**
After this lesson, you should be able to:

- Describe the meaning of "message-level translation"
- Explain how to use translations for single sites and multi-sites
- Identify when a locale would use a manual package release versus an automated package release

**Audience:**

- Translation editors
- Locale managers

# Intro

If you've completed the "How Translations Work" lesson, you now have a general understanding of how the WordPress project handles translation and localization. This includes things like which projects get translated, the priority of translation, and how translation teams are structured.

In this lesson, we will explore more about how translations are used, the technical components of WordPress translation, and how those relate to your responsibilities as a locale manager or translation editor.

# WordPress Internationalization

WordPress' developers chose to use the GNU gettext localization framework to provide localization infrastructure to WordPress. Gettext is a mature, widely used framework for modular translation of software, and is the standard for localization in the open source/free software realm.

Gettext uses message-level translation. Every "message" displayed to users is translated individually, whether it be a paragraph or a single word. In WordPress, such "messages" are generated, translated, and used by the WordPress PHP files via two PHP functions. __() is used when the message is passed as an argument to another function; _e() is used to write the message directly to the page. More detail on these two functions:

__('message')
Searches the localization module for the translation of 'message', and passes the translation to the PHP return statement. If no translation is found for 'message', it just returns 'message'.

_e('message')

Searches the localization module for the translation of 'message', and passes the translation to the PHP echo statement. If no translation is found for 'message', it just echoes 'message'.

There are three types of files used in the gettext translation framework. These files are used and/or generated by translation tools during the translation process, as follows:

**POT (Portable Object Template) files**

The first step in the localization process is that a program is used to search through the WordPress source code and pick out every message passed into a __() or _e() function. This list of English-language messages is put into a specially formatted template file (POT file) that forms the basis of all translations. Generally, you can download a POT file for WordPress, so you shouldn't have to generate your own. Separate POT files can also be made for themes and plugins, if the theme/plugin developer has enclosed all text in __() or _e() functions.

**PO (Portable Object) files**

The second step in the localization process is that the translator translates all the messages from the POT file into the target language, and saves both English and translated messages in a PO file.

**MO (Machine Object) files**

The final step in the localization process is that the PO file is run through a program that turns it into an optimized machine-readable binary file (MO file). Compiling the translations to machine code makes the localized program much faster in retrieving the translations while it is running.

## JSON files

Starting from WordPress 5.0, JavaScript components are translated using JSON files. Similar translation functions as in PHP code exist inside the JavaScript code. These strings are automatically extracted and made available for translation on the translate.wordpress.org site. When a language pack or a localized installation package is created, the translations for JavaScript functions are included as JSON files.

## Direct translation

While GNU gettext works well for most aspects of WordPress, there are a few components of the software that do not lend themselves to localization with gettext or that should not be translated. Those include:

- readme.html: This is a static HTML file, not a PHP file and, as such, can't be run through gettext.
- license.txt: For legal reasons, this should be kept in English. You can, however, add a translated version to the archive.

- wp-config-sample.php

`dist/wp-config-sample.php`:

```
[...]
define('AUTH_KEY',         'вашата супер-ултра-уникална фраза сложете тук');
define('SECURE_AUTH_KEY',  'вашата супер-ултра-уникална фраза сложете тук');
define('LOGGED_IN_KEY',    'вашата супер-ултра-уникална фраза сложете тук');
define('NONCE_KEY',        'вашата супер-ултра-уникална фраза сложете тук');
define('AUTH_SALT',        'вашата супер-ултра-уникална фраза сложете тук');
define('SECURE_AUTH_SALT', 'вашата супер-ултра-уникална фраза сложете тук');
define('LOGGED_IN_SALT',   'вашата супер-ултра-уникална фраза сложете тук');
define('NONCE_SALT',       'вашата супер-ултра-уникална фраза сложете тук');
[...]
```

*An example of how to localize the wp-config-sample.php file.*

### Localization Parameters

The translation projects for each locale contain a few parameters that should never be “translated,” but rather adjusted to the specifics of the target language.

The most important ones are marked as “high prio” and are located in the main core translation and in the “admin” part and can be found by sorting the translation strings in descending priority order.

You can see a full list of these parameters, as well as how or if to localize them, in the Polyglots Handbook page: Working with WordPress Core.

## Releasing WordPress Packages

A release package is a ZIP file consisting of WordPress in its entirety, along with translation files for core, the default themes and Akismet, and any custom changes you may have.

Once your translation of the WordPress core software is complete, it is time to generate a package. Locale managers can create release packages manually, if your locale does not use automated releases.

### Manual release

In this section, we'll go over how to manually release a WordPress package.

First, you need to obtain the revision number of the corresponding WordPress package. Releases are available in the list of tags on core Trac or on this page in the “Release Revisions” section. This list does not include pre-releases including betas and release candidates. For

those, you will need to look for a reference to the version “bump” in the build.trac.wordpress.org revision log.

Once you have the revision number of the version you’ll be packaging, it’s time to build the actual package. To do that, follow these steps:

- Log in as a General Translation Editor to your local WordPress site (locale.wordpress.org)
- Go to “Tools → Release Packages”.
- Scroll down to “Build New Package” section.
- For “Where should we get your translations from”, select “translate.wordpress.org” if you want to use those current translations. If you’ve checked in translated message files in svn, select “Subversion”.
- For “Locale branch for dist files”, select the location of your translated files under https://i18n.svn.wordpress.org/locale/.
- For “WordPress Branch”, select the corresponding original branch.
- For “WordPress revision”, select the revision number you found at the beginning.
- “WordPress Version” will be used as the package name (e.g. 3.9, 3.9-beta3, 3.9-RC1).
- Click the “Build Package” button.

Before releasing the package to the public, download the zip or tar.gz file from the links at the top of the page and test it.

When you’re ready to release the final version to the public, click the version’s “Release” link under the “Action” column. This marks the official release of that version. Users will be prompted for an upgrade of their language package on their dashboard, and download information on your locale site will be updated.

## Automated release

It is recommended that locales and locale managers make sure their locale is eligible for automated release packages. The manual package release process is no longer recommended and subject to removal.

If you have never had any custom changes and i18n.svn.wordpress.org is entirely empty for your locale, you do not need to do anything. Your release package will be created automatically for you.

If you have minor custom changes for the current stable version (like 5.7), consisting of - at most - the following files:

- a translated readme
- license file
- wp-config-sample.php

Please ensure these files exist in a branches/5.7/dist directory at i18n.svn.wordpress.org. If they are, your release package will be created automatically for you.

When is a new package released? A new build will be created only if a translation was changed after the latest build. That means rejecting and approving one translation within the WordPress project on translate.wordpress.org will trigger a new build. Release packages and language packs for stable versions are usually built every hour on the hour.