docs.zendframework.com

Internationalization - tutorials

1. Docs »
2. MVC Tutorials »
3. Internationalization

If you are building a site for an international audience, you will likely want to provide localized versions of common strings on your website, including menu items, form labels, button labels, and more. Additionally, some websites require that route path segments be localized.

Zend Framework provides internationalization (i18n) tools via the zend-i18n component, and integration with zend-mvc via the zend-mvc-i18n component.

Installation

Install zend-mvc-i18n via Composer:

```
$ composer require zendframework/zend-mvc-i18n
```

Assuming you are using zend-component-installer (which is installed by default with the skeleton application), this will prompt you to install the component as a module in your application; make sure you select either `application.config.php` or `modules.config.php` for the location.

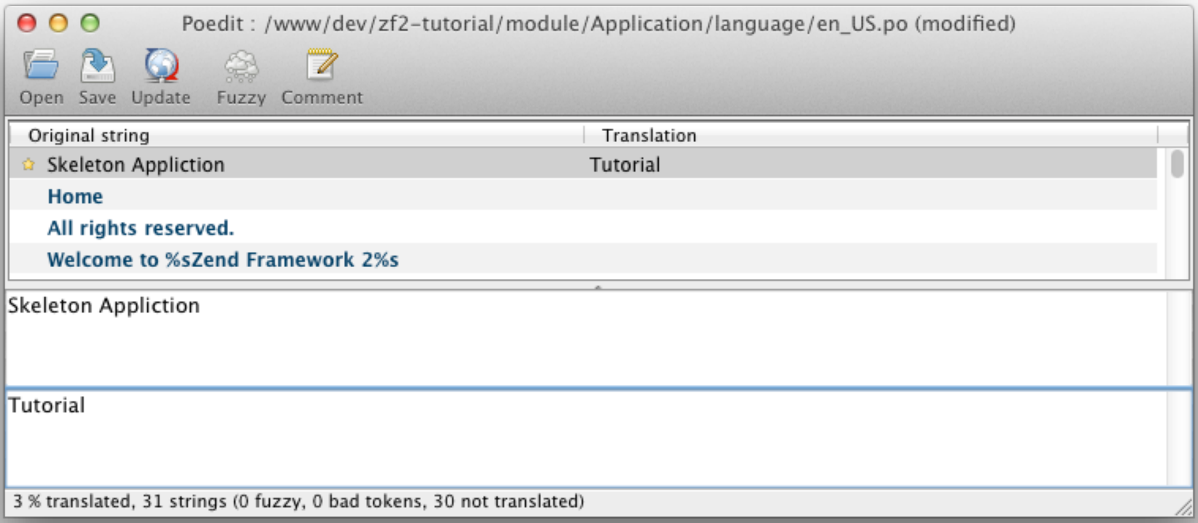
Once installed, this component exposes several services, including:

- `MvcTranslator`, which implements the zend-i18n `TranslatorInterface`, as well as the version specific to zend-validator, providing an instance that can be used for all application contexts.
- A "translator aware" router.

By default, until you configure translations, installation has no practical effect. So the next step is creating translations to use in your application.

Creating translations

The zend-i18n Translation chapter covers the details of adding translations to your application. You can use PHP arrays, INI files, or the popular gettext package (which allows you to use industry standard tools such as poedit to edit translations).


Poedit : /www/dev/zf2-tutorial/module/Application/language/en_US.po (modified)
Open
Save
Update
Fuzzy
Comment

Original string	Translation
Skeleton Appliction	Tutorial
Home	
All rights reserved.	
Welcome to %sZend Framework 2%s	

Skeleton Appliction

Tutorial

3 % translated, 31 strings (0 fuzzy, 0 bad tokens, 30 not translated)

Once you have some translation sources, you will need to put them somewhere your application can access them. Options include:

- In a subdirectory of the module that defines and/or consumes the translation strings. As an example, `module/Application/language/` .
- In your application data directory; e.g., `data/language/` .

Make sure you follow the guidelines from the zend-i18n documentation with regards to naming your files. Additionally, you may want to further segregate any such directory by text domain.

From here, you need to configure the translator to use your files. This requires adding configuration in either your module or application configuration files that provides:

- The default locale if none is provided.
- Translation file patterns, which include:

- the translation source type (e.g., `gettext`, `phparray`, `ini`)
- the base directory in which they are stored
- a file pattern for identifying the files to use

As examples:

```
'translator' => [
    'locale' => 'en_US',
    'translation_file_patterns' => [
        [
            'type'     => 'gettext',
            'base_dir' => __DIR__ . '/../language',
            'pattern'  => '%s.mo',
        ],
    ],
],
```

```
'translator' => [
    'locale' => 'en_US',
    'translation_file_patterns' => [
        [
            'type'     => 'gettext',
            'base_dir' => getcwd() .  '/data/language',
            'pattern'  => '%s.mo',
        ],
    ],
],
```

Once the above configuration is in place, the translator will be active in your application, allowing you to use it.

Translating strings in templates

Once you have defined some strings to translate, and configured the application to use them, you can translate them in your application. The `translate()` and `translatePlural()` view helpers allow you to provide translations within your view scripts.

As an example, you might want to translate the string "All rights reserved" in your footer. You could do the following in your layout script:

```
 <p>&copy; 2016 by Examples Ltd. <?= $this->translate('All rights
reserved') ?></p>
```

Translating route segments

In order to enable route translation, you need to do two things:

- Tell the router to use the translation-aware route class.
- Optionally, tell it which text domain to use (if not using the default text domain).

To tell the application to use the translation-aware route class, we can update our routing configuration. Underneath the top-level `router` key, we'll add the `router_class` key:

```
'router' => [
    'router_class' => Zend\Mvc\I18n\Router
\TranslatorAwareTreeRouteStack::class,
    'routes' => [

    ],
],
```

If you want to use an alternate text domain, you can do so via the `translator_text_domain` key, also directly below the `router` key:

```
'router' => [
    'router_class' => Zend\Mvc\I18n\Router
\TranslatorAwareTreeRouteStack::class,
    'translator_text_domain' => 'router',
    'routes' => [

    ],
],
```

Now that the router is aware of translations, we can use translatable strings in our routes. To do so, surround the string capable of translation with braces (`{}`). As an example:

```
'route' => '/{login}',
```

specifies the word "login" as translatable.