Module translation

Module translation

The module's text strings are written in English, but you might want French, Spanish or Polish shop owners to use your module too. You therefore have to translate those strings into those languages, both the front office and the back office strings. Ideally, you should translate your module in all the languages that are installed on your shop. This could be a tedious task, but a whole system has been put in place in order to help you out.

In short, PrestaShop 1.7 implements Symfony's translation mechanism, through the use of the `trans()` method, used to encapsulate the strings to be translated. This method is applied in a different way depending of the file type.

This mechanism does not work with 3rd-party modules. You should keep using the legacy translation system and it will work just like in v1.6. Otherwise you won't be able to translate your modules, which we want to avoid.

These comments do not apply to theme development, for the whole theme structure has been redesigned in 1.7. 1.7-specific themes should use the new translation system. Just not modules :(

The process of preparing text strings for translation is called internationalization, or i18n.

Internationalizing strings in Smarty (.tpl) files

Strings in TPL files will need to be turned into dynamic content using the `{l}` function call, which Smarty will replace by the translation for the chosen language.

PrestaShop 1.6 used to require the `mod` parameter for context. PrestaShop 1.7 now requires that parameter to be "d", and to use the same domain as all the other strings in the module.

In our sample module, the `mymodule.tpl` file…

```
mymodule.tpl (partial)
<li>
  <a href="{$base_dir}modules/mymodule/mymodule_page.php" title="Click this link">Click me!</a>
</li>
<!-- Block mymodule -->
<div id="mymodule_block_left" class="block">
  <h4>{l s='Welcome!' d='Modules.MyModule'}</h4>
  <div class="block_content">
    <p>Hello,
       {if isset($my_module_name) && $my_module_name}
           {$my_module_name}
       {else}
           World
       {/if}
       !
    </p>
    <ul>
      <li><a href="{$my_module_link}" title="Click this link">Click me!</a></li>
    </ul>
  </div>
</div>
<!-- /Block mymodule -->
```

…becomes:

```
mymodule.tpl (partial)
<li>
  <a href="{$base_dir}modules/mymodule/mymodule_page.php" title="{l s='Click this link' mod='mymodule'}">{l
s='Click me!' mod='mymodule'}</a>
</li>
<!-- Block mymodule -->
<div id="mymodule_block_left" class="block">
  <h4>{l s='Welcome!' d='Modules.MyModule'}</h4>
  <div class="block_content">
    <p>
      {if !isset($my_module_name) || !$my_module_name}
        {capture name='my_module_tempvar'}{l s='World' d='Modules.MyModule'}{/capture}
        {assign var='my_module_name' value=$smarty.capture.my_module_tempvar}
      {/if}
      {l s='Hello %1$s!' sprintf=$my_module_name d='Modules.MyModule'}
    </p>
    <ul>
      <li><a href="{$my_module_link}"  title="{l s='Click this link' d='Modules.MyModule'}">{l s='Click me!'
d='Modules.MyModule'}</a></li>
    </ul>
  </div>
</div>
<!-- /Block mymodule -->
```

…and the `display.tpl` file:

```
display.tpl
Welcome to this page!
```

…becomes:

```
display.tpl
{l s='Welcome to this page!' d='Modules.MyModule'}
```

Notice that we always use the d parameter. This is used by PrestaShop to assert which module the string belongs to. The translation tool needs it in order to match the string to translate with its translation. This parameter is mandatory for module translation.

Internationalizing strings in Twig (.twig) files

To be written.

Generating your translation files

To be written.

Translating your module's strings

Strings are delimited with single quotes. If a string contains single quotes, they should be escaped using a backslash (\).

This way, strings can be directly translated inside PrestaShop:

- Go to the "Translations" page under the "Localization" menu,
- In the "Modify translations" drop-down menu, choose "Installed modules translations",
- Choose the language you want to translate the module into. The destination language must already be installed to enable translation in it.
- Click the "Modify" button.

The page that loads displays all the strings for all the currently-installed modules. Modules that have all their strings already translated have their fieldset closed, whereas if at least one string is missing in a module's translation, its fieldset is expanded. In order to translate your module's strings (the ones that were "marked" using the `l()` method), simply find your module in the list (use the browser's in-page search), and fill the empty fields.

Once all strings for your module are correctly translated, click on either the "Save and stay" button or the "Save" button at the bottom of your list of strings.

PrestaShop then saves the translations in a new file, named using the `languageCode.php` format (for instance, `/mymodule/fr.php`). The translation file looks like so:

```
<?php
global $_MODULE;
$_MODULE = array();
$_MODULE['<{mymodule}prestashop>mymodule_2ddddc2a736e4128ce1cdfd22b041e7f'] = 'Mon module';
$_MODULE['<{mymodule}prestashop>mymodule_d6968577f69f08c93c209bd8b6b3d4d5'] = 'Description du module.';
$_MODULE['<{mymodule}prestashop>mymodule_533937acf0e84c92e787614bbb16a7a0'] = 'Êtes-vous certain de vouloir
désinstaller ce module ? Vous perdrez tous vos réglages !';
$_MODULE['<{mymodule}prestashop>mymodule_0f40e8817b005044250943f57a21c5e7'] = 'Aucun nom fourni';
$_MODULE['<{mymodule}prestashop>mymodule_fe5d926454b6a8144efce13a44d019ba'] = 'Valeur de configuration non
valide.';
$_MODULE['<{mymodule}prestashop>mymodule_c888438d14855d7d96a2724ee9c306bd'] = 'Réglages mis à jour';
$_MODULE['<{mymodule}prestashop>mymodule_f4f70727dc34561dfde1a3c529b6205c'] = 'Réglages';
$_MODULE['<{mymodule}prestashop>mymodule_2f6e771db304264c8104cb7534bb80cd'] = 'Valeur de configuration';
$_MODULE['<{mymodule}prestashop>mymodule_c9cc8cce247e49bae79f15173ce97354'] = 'Enregistrer';
$_MODULE['<{mymodule}prestashop>mymodule_630f6dc397fe74e52d5189e2c80f282b'] = 'Retour à la liste';
$_MODULE['<{mymodule}prestashop>display_86e88cbccafa83831b4c6685501c6e58'] = 'Bienvenue sur cette page !';
$_MODULE['<{mymodule}prestashop>mymodule_9a843f20677a52ca79af903123147af0'] = 'Bienvenue !';
$_MODULE['<{mymodule}prestashop>mymodule_f5a7924e621e84c9280a9a27e1bcb7f6'] = 'Monde';
$_MODULE['<{mymodule}prestashop>mymodule_3af204e311ba60e6556822eac1437208'] = 'Bonjour %s !';
$_MODULE['<{mymodule}prestashop>mymodule_c66b10fbf9cb6526d0f7d7a602a09b75'] = 'Cliquez sur ce lien';
$_MODULE['<{mymodule}prestashop>mymodule_f42c5e677c97b2167e7e6b1e0028ec6d'] = 'Cliquez-moi !';
```

This file must not be edited manually! It can only be edited through the PrestaShop translation tool.

Now that we have a French translation, we can click on the French flag in the front office, and get the expected result: the module's strings are now in French.

They are also translated in French when the back office is in French.

Translating complex code

As we can see, the basis of template file translation is to enclose them in the `{l s='The string' mod='name_of_the_module'}`. The changes in `display.tpl` and in `mymodule.tpl`'s link and title texts are thus easy to understand. But added a trickier block of code for the "Hello World!" string: an if /else/then clause, and a text variable. Let's explore this code:

Here is the original code:

```
Hello,
  {if isset($my_module_name) && $my_module_name}
    {$my_module_name}
  {else}
    World
  {/if}
!
```

As you can see, we need to get the “Hello World” string translatable, but also to cater for the fact that there is a variable. As explained in the “Translations in PrestaShop 1.5” chapter, variables are to be marked using `sprintf()` markers, such as `%s` or `%1$s`.

Making “Hello %s!” translatable words in easy: we just need to use this code:

```
{l s='Hello %s!' sprintf=$my_module_name d='Modules.MyModule'}
```

But in our case, we also need to make sure that the %s is replaced by “World” in case the “my_module_name” value does not exist… and we must make “World” translatable too. This can be achieved by using Smarty `{capture}` function, which collects the output of the template between the tags into a variable instead of displaying, so that we can use it later on. We are going to use it in order to replace the variable with the translated “World” if the variable is empty or absent, using a temporary variable.

Here is the final code:

```
{if !isset($my_module_name) || !$my_module_name}
  {capture name='my_module_tempvar'}{l s='World' d='Modules.MyModule'}{/capture}
  {assign var='my_module_name' value=$smarty.capture.my_module_tempvar}
{/if}
{l s='Hello %s!' sprintf=$my_module_name d='Modules.MyModule'}
```