Créer un module PrestaShop

Table des matières

Créer un module PrestaShop

Principes opératoires des modules

Les modules sont la meilleure manière de laisser votre talent de développeur et votre imagination s'exprimer, tant les possibilités créatives sont nombreuses.

Ils peuvent afficher une grande variété de contenus (blocs, texte, etc.), réaliser de nombreuses tâches (mise à jour groupées, import, export, etc.), créer une liaison avec d'autres outils...

Les modules peuvent être aussi configurables que nécessaire ; plus ils le sont, plus ils seront utiles, et donc capables de répondre aux besoins d'un plus grand nombre d'utilisateurs.

L'un des principaux intérêts des modules et d'ajouter des fonctionnalités à PrestaShop sans devoir modifier ses fichiers internes, rendant possible le fait de mettre la solution à jour sans devoir recopier toutes ses modifications.

De fait, vous devriez toujours éviter de toucher au fichier interne de PrestaShop lorsque vous concevez un module, même si cela peut être difficile dans certaines situations...

Arborescence des fichiers du module

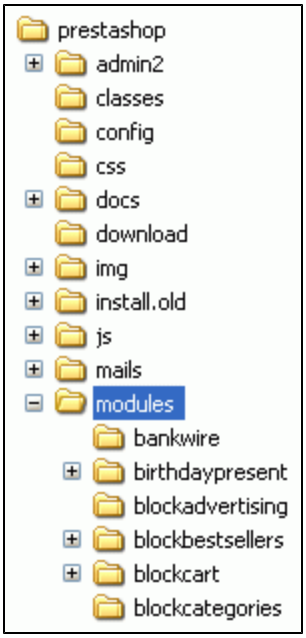
Tous les modules PrestaShop sont installés dans le dossier `/modules`, qui se trouve à la racine du dossier principal de PrestaShop. Cela s'applique autant aux modules par défaut (ceux fournis avec PrestaShop) qu'aux modules tiers que vous pourriez installer par la suite.

Chaque module dispose de son propre sous-dossier dans le dossier `/modules` : `/bankwire`, `/birthdaypresent`, etc.

Structure de base d'un module

Tous les modules utilisent la même structure de base, ce qui facilite l'apprentissage en regardant le code source de chacun.

Créons un premier module très simple ; celui nous permettra de mieux en décrire la structure. Nous le nommerons "My module".



Créons tout d'abord le dossier du module. Il devrait avoir le même nom que le module, avec aucune espace, et uniquement des caractères alphanumériques, le tiret "-" et le caractère souligné "_", le tout en minuscule : `/mymodule`.

Ce dossier doit contenir un fichier PHP du même nom, qui s'occupera de la plupart des traitements: `mymodule.php`.

C'est là la base pour un module très simple, mais bien entendu il est possible d'ajouter d'autres fichiers et dossiers.

La partie publique du module doit être définie dans un fichier `.tpl` placé à la racine du dossier du module. Les fichiers TPL peuvent prendre n'importe quel nom, s'il n'y en a qu'un seul, une bonne pratique consiste à lui donner le même nom que le dossier et le fichier principal : `mymodule.tpl`.

Ce fichier `mymondule.php` doit commencer avec le test suivant:

```
if (!defined('_PS_VERSION_'))
  exit;
```

Celui-ci vérifie l'existence d'une constante PHP, et quitte si elle n'existe pas. Le seul but de ce test est d'empêcher les visiteurs d'accéder directement ce fichier.

Ce fichier doit également contenir la classe du module. PrestaShop utilise la programmation orientée Objet, et de fait ses modules également.

Cette classe doit porter le même nom que le module et son dossier, en CamelCase : `MyModule`.
Qui plus est, cette classe doit étendre la classe `Module`, et donc hérite de toutes ses méthodes et attributs. Elle peut tout aussi bien étendre n'importe quelle classe dérivée de la classe `Module` : `PaymentModule`, `ModuleGridEngine`, `ModuleGraph`...

mymodule.php

```
<?php
if (!defined('_PS_VERSION_'))
  exit;

class MyModule extends Module
  {
  public function __construct()
    {
    $this->name = 'mymodule';
    $this->tab = 'Test';
    $this->version = 1.0;
    $this->author = 'Firstname Lastname';
    $this->need_instance = 0;

    parent::__construct();

    $this->displayName = $this->l('My module');
    $this->description = $this->l('Description of my module.');
    }

  public function install()
    {
    if (parent::install() == false)
      return false;
    return true;
    }
  }
?>
```

Examinons chaque ligne de l'objet `MyModule`...

```
public function __construct()
```

Définit le constructeur de la classe.

```
$this->name = 'mymodule';
$this->tab = 'Test';
$this->version = 1.0;
$this->author = 'PrestaShop';
```

Cette section assigne une poignée d'attributs à l'instance de classe `this` :

- Un attribut 'name'. Il s'agit d'un identifiant interne, donc il est préférable de s'assurer qu'il est unique, sans caractères spéciaux ni espaces, et de le garder en minuscule.
- Un attribut 'tab'. C'est le nom du tableau qui contiendra ce module dans la liste des modules du back-office de PrestaShop. Vous pouvez utiliser un nom existant, comme `Products`, `Blocks` ou `Stats`, ou en choisir un personnalisé, comme nous l'avons fait ici. Dans ce dernier cas, un nouveau tableau sera ajouté avec votre titre.
- Un numéro de version pour le module, qui est affiché dans la liste de modules.
- Un attribut 'author'. Le nom de l'auteur est affiché dans la liste de modules de PrestaShop.

```
$this->need_instance = 0;
```

Le drapeau `need_instance` indique s'il faut oui ou non charger la classe du module lors du chargement de la page "Modules" dans le back-office. S'il est à 0, le module n'est pas chargé, et donc la page des modules utilisera moins de ressources. Si vos modules ont besoin d'afficher un avertissement dans la page des modules, alors vous devez mettre cet attribut à 1.

```
parent::__construct();
```

Appelle le constructeur du parent. Cela doit être fait avant tout appel à la méthode `$this->l()`, et après avoir créé `$this->name`.

```
$this->displayName = $this->l('My module');
```

Assigne un nom public au module, nom qui sera affiché dans la liste des modules, dans le back-office. La méthode `l()` fait partie des outils de traduction de PrestaShop, et est expliquée plus bas.

```
$this->description = $this->l('Description of my module.');
```

Assigne une description publique pour le module, qui sera affichée dans la liste des modules.

```
public function install()
  {
  return (parent::install());
  }
```

Sous cette première incarnation extrêmement simple, cette méthode est inutile, étant donné que tout ce qu'elle fait est vérifier la valeur renvoyée par la méthode `install()` de la classe `Module`. Par ailleurs, si nous n'avions pas créé cette méthode, la méthode de la superclasse aurait été appelée de toute façon, amenant au même résultat.
Cependant, nous devons mentionner cette méthode, car elle nous sera très utile une fois que nous aurons à réaliser des tests et des actions lors du processus d'installation du module : créer des tables SQL, copier des fichiers, créer des variables de configuration, etc.

De la même manière, le module devrait contenir une méthode `uninstall()`, afin de disposer d'un processus de désinstallation personnalisé. Cette méthode pourrait être comme suit :

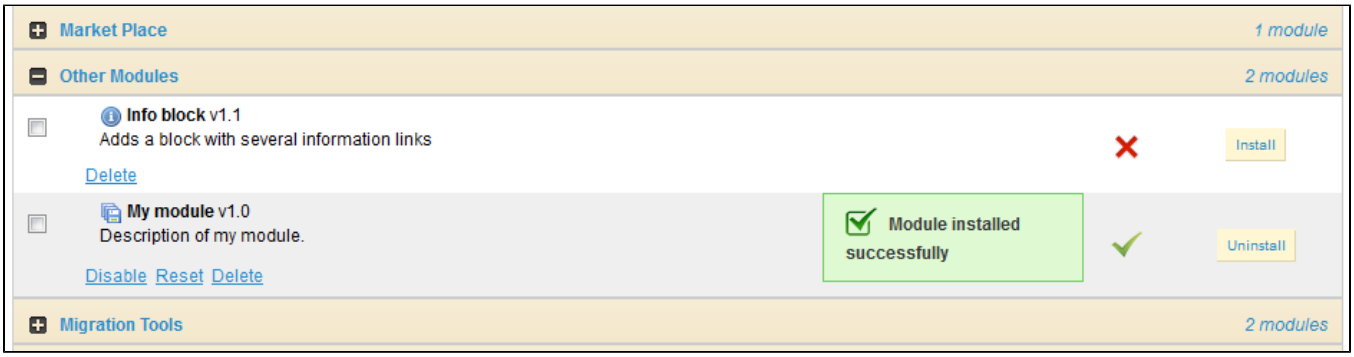
```
public function uninstall()
  {
  if (!parent::uninstall())
    Db::getInstance()->Execute('DELETE FROM `'._DB_PREFIX_.'mymodule`');
  parent::uninstall();
  }
```

Pour parfaire ce premier module, nous pouvons ajouter une icône, qui sera affiché à côté du nom du module dans la liste des modules.
Le fichier d'icône doit respecter le format suivant :

- image en 16*16 pixels ;
- nommée `logo.gif` ;
- placée dans le dossier principal du module.

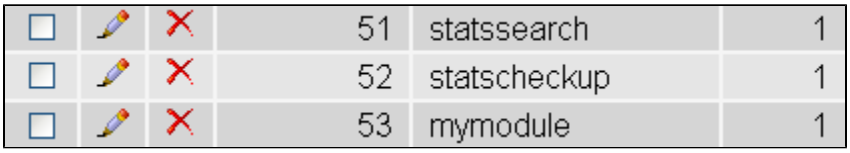
Vous trouverez un excellent jeu gratuit d'icônes sur le site FamFamFam.

Maintenant que toutes les bases sont en place, mettez le dossier du modules dans le dossier `/modules` de votre installation test de PrestaShop, ouvrez PrestaShop, et dans l'onglet "Modules", sous "Autres Modules", vous devriez trouver votre module. Installez-le afin de pouvoir le gérer pour la suite de ce guide.



PrestaShop crée automatiquement un petit fichier `config.xml` dans le dossier du module, fichier qui stocke certaines informations de configuration. Vous ne devriez JAMAIS le modifier manuellement.

Lors de l'installation, PrestaShop ajoute également une ligne à la table SQL `ps_module`.



Accrocher un module

Afficher des données, lancer un processus à une heure donnée : afin "d'attacher" un module à un emplacement du front-office ou du back-office, vous devez lui donner accès à l'un des nombreux points d'accroche de PrestaShop, décrits plus avant dans ce guide.

Pour ce faire, nous allons changer le code de notre module, et ajouter ces lignes :

mymodule.php (partial)

```
public function install()
  {
  if (parent::install() == false OR !$this->registerHook('leftColumn'))
    return false;
  return true;
  }

...

public function hookLeftColumn($params)
  {
  global $smarty;
  return $this->display(__FILE__, 'mymodule.tpl');
  }

public function hookRightColumn($params)
  {
  return $this->hookLeftColumn($params);
  }
```

Explorons les lignes ajoutées/modifiées :

```
if (parent::install() == false OR !$this->registerHook('leftColumn'))
  return false;
return true;
```

Nous avons modifié la ligne originale pour faire un second test.

Ce code vérifie :

- la valeur booléenne renvoyée par la méthode `install()` de la classe `Module` : si elle est `true`, alors le module est installé et peut être utilisé.
- la valeur booléenne renvoyée par la méthode `registerHook()` pour le point d'accroche `leftColumn` : si elle est `true`, alors le module est en enregistré pour le point d'accroche dont il a besoin, et peut être utilisé.

Il suffit que l'une de ces deux valeurs soit `false` pour que `install()` renvoie `false` également, et que le module ne puisse être installé. Les deux valeurs doivent être `true` pour que le module soit installé.

De fait, cette ligne peut se lire comme suit : si l'installation ou l'accrochage échouent, nous en informons PrestaShop.

```
public function hookLeftColumn($params)
  {
  global $smarty;
  return $this->display(__FILE__, 'mymodule.tpl');
  }
```

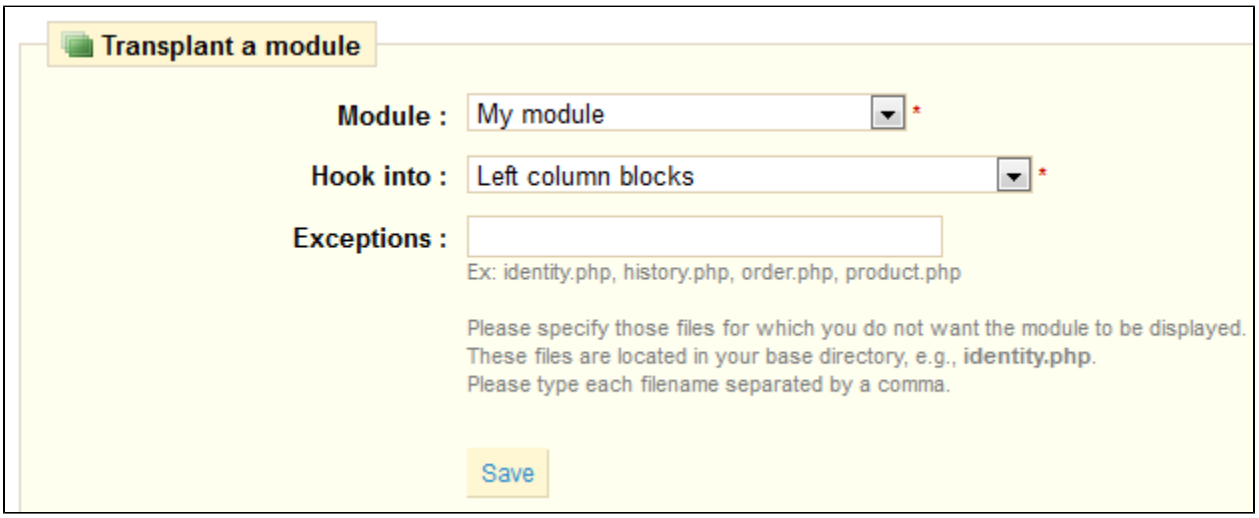
La méthode `hookLeftColumn()` fait en sorte que le module puisse s'accrocher au point d'accroche de la colonne de gauche du thème.
`$smarty` est la variable globale du système de modèle Smarty, utilisé par PrestaShop, et à laquelle nous devons accéder.
La méthode `display()` renvoie le contenu du fichier de template `mymodule.tpl`, s'il existe.

```
public function hookRightColumn($params)
  {
  return $this->hookLeftColumn($params);
  }
```

De la même manière, `hookRightColumn()` donne accès au thème de la colonne de droite. Dans cet exemple, nous appelons simplement la méthode `hookLeftColumn()` afin d'obtenir le même affichage, quelle que soit la colonne.

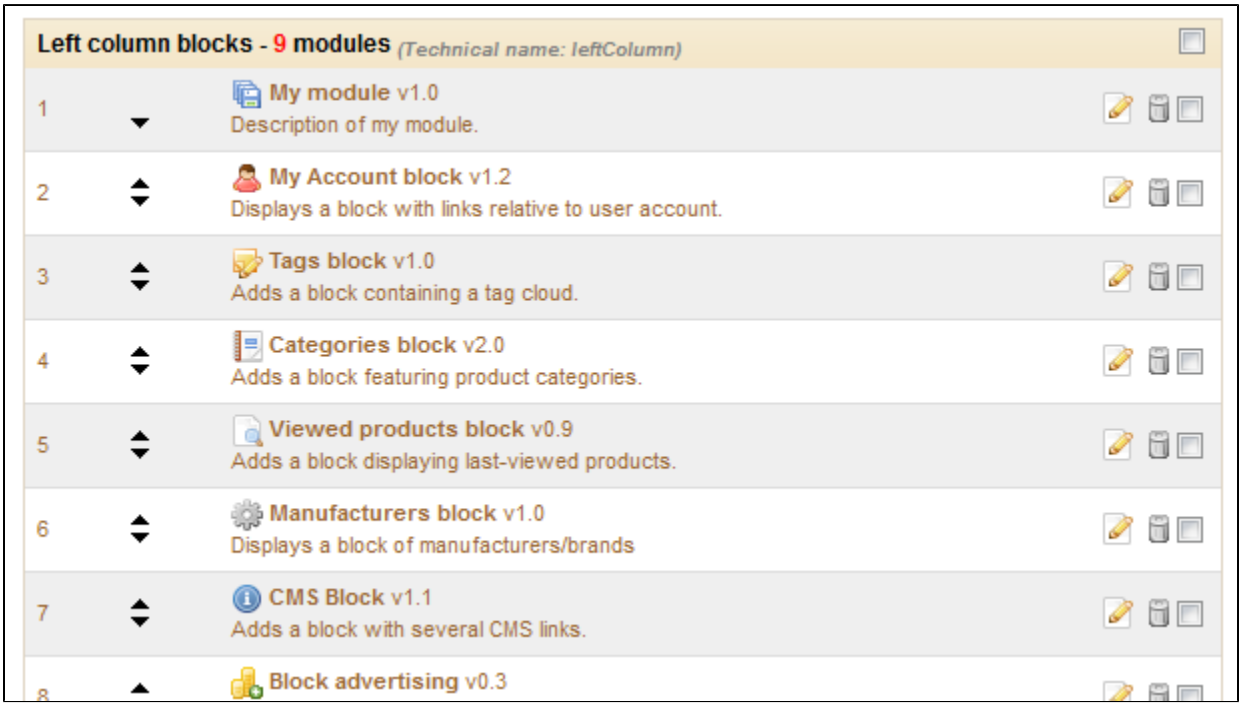
Enregistrez le fichier, et vous pouvez d'ores et déjà l'accrocher au thème, le déplacer et le greffer : aller au sous-onglet "Positions" de l'onglet "Module" du back-office, puis cliquer sur le lien "Greffer un module".

Dans le formulaire de greffe, trouvez "My module" dans le menu déroulant de modules, puis choisissez "Left menu blocks" dans le menu déroulant "Greffer le module sur".



Inutile d'essayer d'accrocher un module à un point d'accroche pour lequel il n'implémente aucune méthode.

Enregistrez. La page "Positions" devrait se recharger, avec le message suivant : "Le module a bien été greffé au hook". Félicitations ! Descendez dans la page, et vous devriez effectivement voir votre module parmi les autres modules dans la liste "Left column blocks". Déplacez-le en haut de la liste.



Affiche du contenu

Maintenant que nous avons accès à la colonne de gauche, nous pouvons y afficher quelque chose.

Comme indiqué plus tôt, le contenu à afficher dans le thème doit être stocké dans des fichiers `.tpl`. Nous allons créer le fichier `mymodule.tpl`, qui a été passé comme paramètre à la méthode `display()` du code de notre module.

Créons donc le fichier `mymodule.tpl`, et ajoutons-lui quelques lignes de code.

mymodule.tpl

```
<!-- Block mymodule -->
<div id="mymodule_block_left" class="block">
  <h4>Welcome!</h4>
  <div class="block_content">
    <ul>
      <li><a href="{$base_dir}modules/mymodule/mymodule_page.php" title="Click this link">Click me!</a></li>
    </ul>
  </div>
</div>
<!-- /Block mymodule -->
```

Enregistrez le fichier dans le dossier racine du module, et rechargez la page d'accueil de la boutique : il devrait apparaître en haut de la colonne de gauche, juste à côté du logo de la boutique.

Le lien affiché ne mène à rien pour le moment. Si vous avez besoin de le tester, ajoutez le fichier `mymodule_page.php` dans le dossier du module, avec un contenu minimal, tel qu'un simple "Bienvenu dans ma boutique !" La page résultante sera brute d'aspect, donc nous allons voir comment lui appliquer le style du thème.

Comme vous pouvez vous y attendre, nous devons créer un fichier TPL pour pouvoir exploiter le style du thème. Créons donc le fichier `mymodule_page.tpl`, qui contiendra notre message basique, et appelons ce fichier depuis `mymodule_page.php`, qui ajoutera le thème (en-tête, pied de page, etc.).

Vous devez vous efforcer d'utiliser des noms explicites et facilement reconnaissables pour vos fichiers TPL, afin de les trouver facilement dans le back-office – ce qui est particulièrement important lors de l'utilisation de l'outil interne de traduction.

mymodule_page.tpl

```
Welcome to my shop!
```

mymodule_page.php

```
<?php
global $smarty;
include('../../config/config.inc.php');
include(../../header.php');

$smarty->display(dirname(__FILE__).'/mymodule_page.tpl');

include('../../footer.php');
?>
```

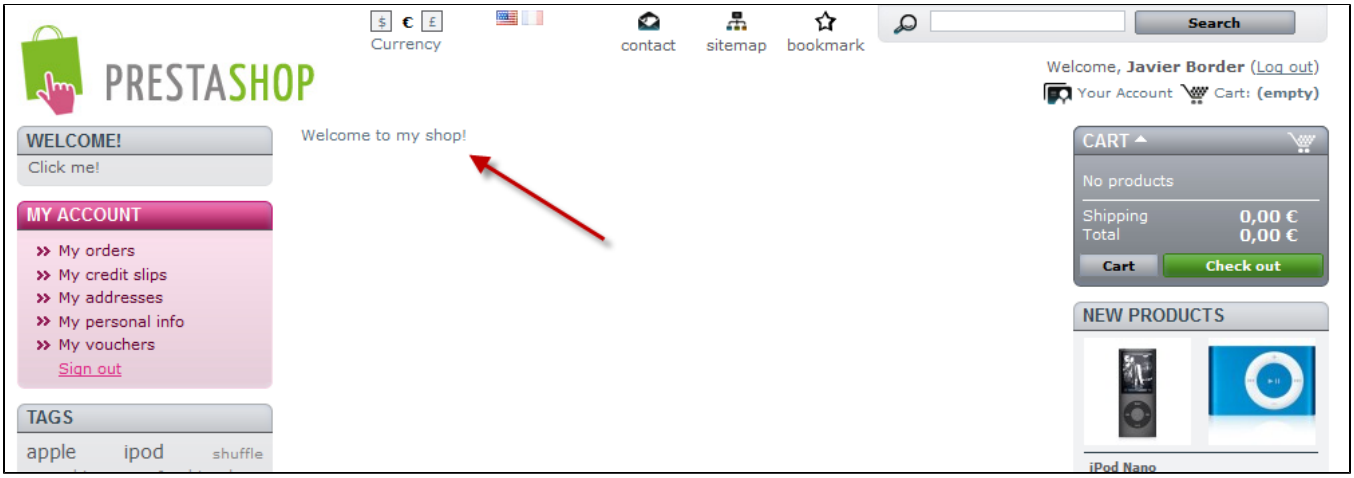
Nous chargeons en premier l'instance actuelle de Smarty. Cela doit impérativement être fait avant l'appel à la méthode `display()`.

Les divers appels `includes()` du fichier nous permettent de charger :

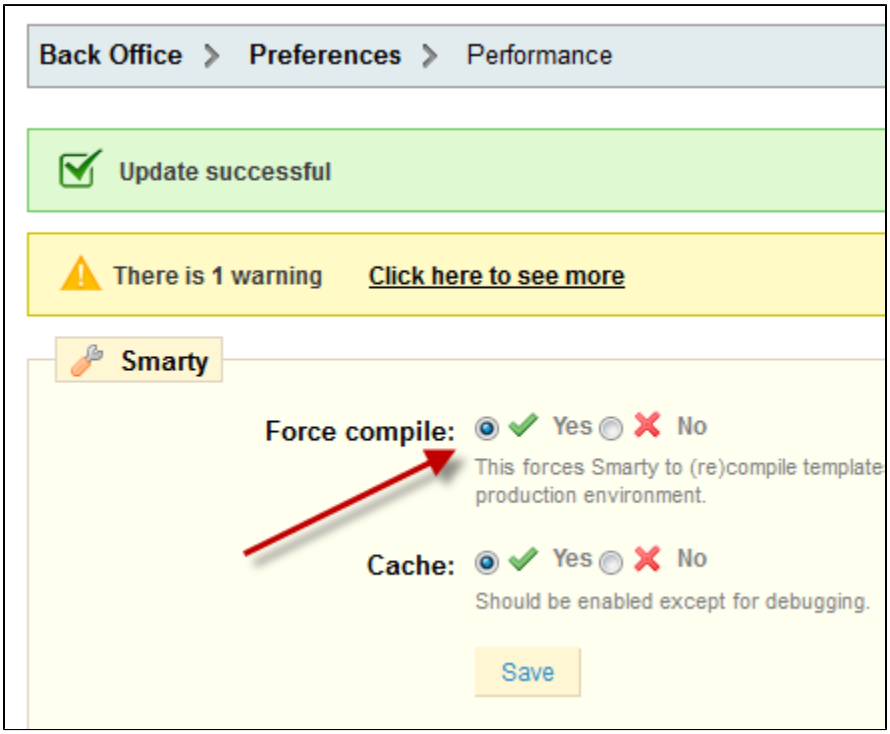
- La configuration actuelle de PrestaShop ;
- le fichier de l'en-tête du thème (via `header.php`, qui agit comme fichier de chargement).
- le fichier de pied de page du thème (via `footer.php`, qui agit comme fichier de chargement).

Au milieu de tout cela, nous plaçons notre fichier TPL personnalisé, dont la seule action sera d'afficher la ligne "Bienvenu dans ma boutique !"

Enregistrez tous les fichiers et rechargez la page d'accueil de votre boutique : en quelques lignes, le résultat final a été nettement amélioré, avec notre message maintenant correctement placé entre en-tête, pied de page et colonnes !



Si vous faites de nombreuses modifications et autant de rechargements de votre page d'accueil, il peut arriver que ces modifications ne s'appliquent pas. La cause de ceci est le cache de Smarty, qui conserve une version compilée de la page d'accueil. Pour forcer Smarty à recompiler les modèles à chaque chargement, vous devez vous rendre dans l'onglet "Préférences", dans son sous-onglet "Performances", et choisir "Oui" pour l'option "Forcer la compilation".



Ne forcez pas la compilation sur les sites en production, car cela ralentit sévèrement celui-ci !

Utiliser Smarty

Smarty est un moteur de modèle/template en PHP, et est utilisé par PrestaShop pour son système de thème.

Il parcourt les fichiers TPL, à la recherche d'éléments dynamiques à remplacer par les données équivalentes, puis affiché le résultat ainsi produit. Ces éléments dynamiques sont indiqués avec des accolades : `{ ... }`. Le programmeur peut créer de nouvelles variables et les utiliser dans ses fichiers TPL.

Par exemple, dans notre `mymodule_page.php`, nous pouvons créer une telle variable :

mymodule_page.php

```
<?php
global $smarty;

include('../../config/config.inc.php');
include('../../header.php');

$mymodule = new MyModule();
$message = $mymodule->l('Welcome to my shop!');
$smarty->assign('messageSmarty', $message ); // creation of our variable
$smarty->display(dirname(__FILE__).'/mymodule_page.tpl');

include( '../../footer.php' );
?>
```

De là, nous pouvons demander à Smarty d'afficher le contenu de cette variable dans notre fichier TPL.

mymodule_page.tpl

```
{$messageSmarty}
```

PrestaShop comprend un certain nombre de variables. Par exemple {$HOOK_LEFT_COLUMN} sera remplacé par le contenu de la colonne de gauche, et donc le contenu de tous les modules qui ont été attachés au point d'accroche de la colonne de gauche.

Toutes les variables Smarty sont globales. Vous devriez donc faire attention à ne pas donner à vos variables un nom déjà utilisé par une autre variable Smarty, afin d'éviter les conflits et les réécritures. Une bonne pratique consiste à éviter les noms trop simples, comme `products`, mais à préfixer du nom de votre module, voire de votre nom. Donc : `{$mark_mymodule_product}`.

Voici une liste de variables Smarty qui sont accessibles depuis toutes les pages :

Fichier / Dossier	Description
img_ps_dir	URL for the PrestaShop image folder.
img_cat_dir	URL for the categories images folder.
img_lang_dir	URL for the languages images folder.
img_prod_dir	URL for the products images folder.
img_manu_dir	URL for the manufacturers images folder.
img_sup_dir	URL for the suppliers images folder.
img_ship_dir	URL for the carriers (shipping) images folder.
img_dir	URL for the theme's images folder.
css_dir	URL for the theme's CSS folder.
js_dir	URL for the theme's JavaScript folder.
tpl_dir	URL for the current theme's folder.
modules_dir	URL the modules folder.

mail_dir	URL for the mail templates folder.
pic_dir	URL for the pictures upload folder.
lang_iso	ISO code for the current language.
come_from	URL for the visitor's origin.
shop_name	Shop name.
cart_qties	Number of products in the cart.
cart	The cart.
currencies	The various available currencies.
id_currency_cookie	ID of the current currency.
currency	Currency object (currently used currency).
cookie	User cookie.
languages	The various available languages.
logged	Indicates whether the visitor is logged to a customer account.
page_name	Page name.
customerName	Client name (if logged in).
priceDisplay	Price display method (with or without taxes...).
roundMode	Rounding method in use.
use_taxes	Indicates whether taxes are enabled or not.

Si vous avez besoin d'afficher toutes les variables Smarty de la page, ajoutez la fonction suivante :

```
{debug}
```

Les commentaires sont formés avec un astérisque.

```
{* Cette ligne est commentée. *}

{*
Cette ligne également !
*}
```

A la différence des commentaires HTML, le code Smarty commenté n'apparaît pas dans le fichier final.

Traduction d'un module

Les chaînes de texte de notre module sont écrites en anglais, mais nous voudrons sans doute que les français puissent également utiliser notre module. Nous devons donc traduire ces chaînes en français, à la fois celles du front-office et celles du back-office. Cela pourrait être une tâche laborieuse, mais Smarty et les propres outils de traduction de PrestaShop facilitent cela.

Les chaînes des fichiers PHP devront être affichées par le biais de la méthode `l()`, provenant de la classe abstraite `Module.php`.

mymodule.php (partial)

```
...
$this->displayName = $this->l('My module');
$this->description = $this->l('Description of my module.');
...
```

Les chaînes des fichiers TPL devront être transformées en contenu dynamique, que Smarty remplacera par la traduction dans la langue choisie. Dans notre module d'exemple, ce fichier :

mymodule.tpl (partial)

```
<li>
  <a href="{$base_dir}modules/mymodule/mymodule_page.php" title="Click this link">Click me!</a>
</li>
```

...devient :

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>
```

...et celui-ci:

mymodule_page.tpl

```
<h4>Welcome!</h4>
...
Click me!
```

...devient :

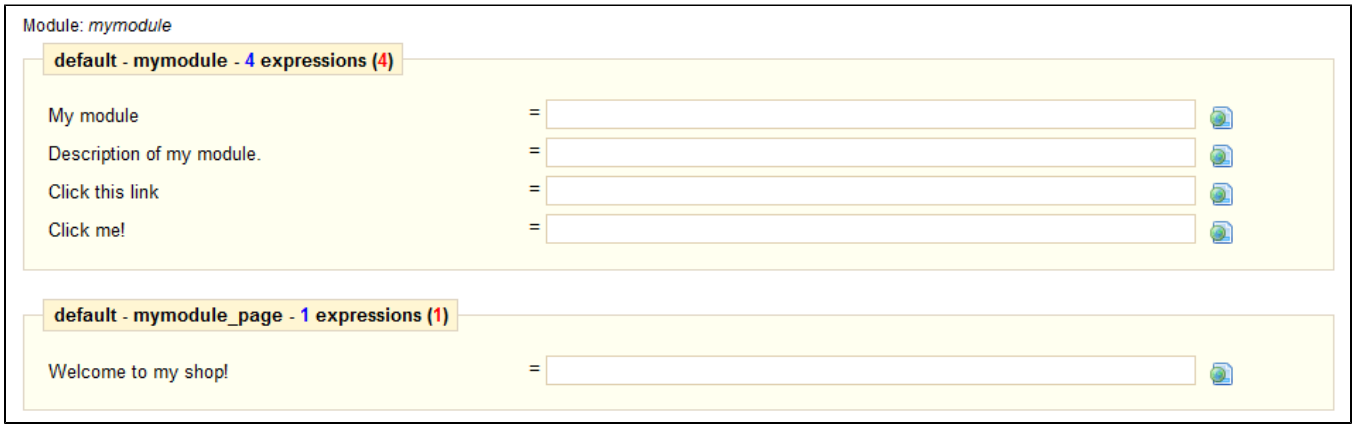
mymodule.tpl

```
<h4>{l s='Welcome!' mod='mymodule'}</h4>
...
{l s='Click me!' mod='mymodule'}
```

L'outil de traduction a besoin du paramètre `mod` pour faire la correspondance entre les chaînes et leurs traductions.
Les chaînes sont délimitées par des apostrophes. Si une chaîne contient des guillemets, ils devront être échappés à l'aide d'un backlash : `\"`.

Ainsi, les chaînes peuvent être directement traduites dans PrestaShop : allez dans l'onglet "Outils", son sous-onglet "Traductions", et dans le menu déroulant "Modifier les traductions", choisissez "Traductions de module", puis cliquez sur le drapeau français pour commencer à traduire les modules en Français.

La page suivante affichera toutes les chaînes de tous les modules actuellement installés. Les modules dont toutes les chaînes sont déjà traduites ont leur bloc de chaînes replié, tandis que ceux ayant au moins une chaîne manquante ont le leur déplié. Afin de traduire les chaînes de votre module (celles qui ont été "marquées" avec la méthode `l()`), trouvez simplement votre module dans la liste (utiliser la recherche de votre navigateur), et remplissez les champs vides.



Une fois que toutes les chaînes sont correctement traduites, cliquez sur le bouton "Mettre à jour la traduction", qui se trouve en haut et en bas de la page.

Chaque champ dispose d'une icône à sa droite. Elle vous permet d'obtenir une suggestion depuis Google Translate. Vous pouvez le survoler avec votre souris pour voir la suggestion, et cliquer sur l'icône pour appliquer la suggestion au champ.

Les traductions automatiques ne sont pas toujours précises, vérifiez bien ce qui vous est proposé.

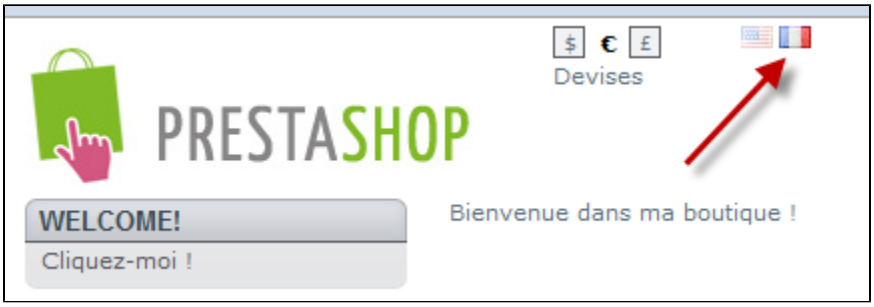
Les traductions sont enregistrées dans un nouveau fichier, `fr.php` (ou plus globalement, `code-de-la-langue.php`, qui est généré par PrestaShop et ressemble à ceci :

mymodule.tpl

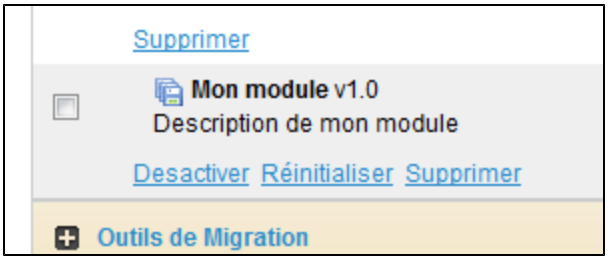
```
<?php

global $_MODULE;
$_MODULE = array();
$_MODULE['<{mymodule}prestashop>mymodule_2ddddc2a736e4128ce1cdfd22b041e7f'] = 'Mon module';
$_MODULE['<{mymodule}prestashop>mymodule_d6968577f69f08c93c209bd8b6b3d4d5'] = 'Description de mon module';
$_MODULE['<{mymodule}prestashop>mymodule_c66b10fbf9cb6526d0f7d7a602a09b75'] = 'Cliquez sur ce lien';
$_MODULE['<{mymodule}prestashop>mymodule_f42c5e677c97b2167e7e6b1e0028ec6d'] = 'Cliquez-moi \!';
$_MODULE['<{mymodule}prestashop>mymodule_page_c0d7cffa0105851272f83d5c1fe63a1c'] = 'Bienvenue dans ma boutique
\!';
```

Ce fichier ne **doit pas** être modifié manuellement ! Il ne doit être modifié que via l'outil de traduction de PrestaShop.

Maintenant que vous disposez d'une traduction, vous pouvez cliquer sur le drapeau français dans le front-office (pourvu que vous ayez effectivement activé la langue), et obtenir le résultat suivant : un module traduit en français.



Les chaînes sont également traduites en français quand le back-office est en français.



Pour que les chaînes à traduire soient prises en compte par l'outil de traduction de PrestaShop, il faut que les fichiers PHP et TPL soient à la racine du dossier du module.

Créer l'onglet d'administration du module, et sa classe

Dans cette section, nous allons voir comment donner à votre module son propre onglet ou sous-onglet, en quelques minutes.

Suivez les étapes suivantes :

1. Ajoutez une nouvelle table à votre base de données PrestaShop, nommée `ps_test`. Donnez-lui deux champs :
    - `id_test` (INT 11) ;
    - `test` (VARCHAT 32).
2. Créez un fichier vide nommé `Test.php` dans le dossier `/classes` de PrestaShop ;
3. Ajoutez les lignes suivantes à ce fichier :

Test.php

```
<?php
class Test extends ObjectModel
  {
  /** @var string Name */
  public $test;

  protected $fieldsRequired = array('test');
  protected $fieldsSize = array('test' => 64);
  protected $fieldsValidate = array('test' => 'isGenericName');
  protected $table = 'test';
  protected $identifier = 'id_test';

  public function getFields()
    {
    parent::validateFields();
    $fields['test'] = pSQL($this->test);
    return $fields;
    }
  }
?>
```

1. Créez un fichier vide nommé `AdminTest.php`, dans le dossier `/admin/tabs` de PrestaShop ;
2. Ajouter les lignes suivantes à ce fichier :

AdminTest.php

```
<?php
include_once(PS_ADMIN_DIR.'/../classes/AdminTab.php');

class AdminTest extends AdminTab
  {
  public function __construct()
    {
    $this->table = 'test';
    $this->className = 'Test';
    $this->lang = false;
    $this->edit = true;
    $this->delete = true;
    $this->fieldsDisplay = array(
      'id_test' => array(
        'title' => $this->l('ID'),
        'align' => 'center',
        'width' => 25),
      'test' => array(
        'title' => $this->l('Name'),
        'width' => 200)
    );

    $this->identifier = 'id_test';

    parent::__construct();
    }

  public function displayForm()
    {
    global $currentIndex;

    $defaultLanguage = intval(Configuration::get('PS_LANG_DEFAULT'));
    $languages = Language::getLanguages();
    $obj = $this->loadObject(true);

    echo '
      <script type="text/javascript">
        id_language = Number('.$defaultLanguage.');
      </script>';

    echo '
      <form action="' . $currentIndex . '&submitAdd' .  $this->table . '=1&token=' . $this->token . '" method="
post" class="width3">
        ' . ($obj->id ? '<input type="hidden" name="id_' . $this->table . '" value="' . $obj->id . '" />' :
'').'
        <fieldset><legend><img src="../img/admin/profiles.png" />' . $this->l('Profiles') . '</legend>
          <label>'.$this->l('Name:').' </label>
          <div class="margin-form">';
    foreach ( $languages as $language )
      echo '
          <div id="name_' . $language['id_lang'|'id_lang'] . '" style="display: ' . ($language
['id_lang'|'id_lang'] == $defaultLanguage ? 'block' : 'none') . '; float: left;">
            <input size="33" type="text" name="name_' . $language['id_lang'|'id_lang'] . '" value="' .
htmlentities( $this->getFieldValue( $obj, 'name', intval( $language['id_lang'|'id_lang'] ) ), ENT_COMPAT, 'UTF-
8' ) . '" /><sup>*</sup>
          </div>';
    $this->displayFlags( $languages, $defaultLanguage, 'name', 'name' );
    echo '
          <div class="clear"></div>
        </div>
        <div class="margin-form">
          <input type="submit" value="'.$this->l('Save').'" name="submitAdd'.$this->table.'" class="button" />
        </div>
        <div class="small"><sup>*</sup> '.$this->l('Required field').'</div>
      </fieldset>
    </form> ';
    }
  }
?>
```

Mettez ces fichiers en ligne, puis créez l'onglet en vous rendant dans l'onglet "Employés", puis son sous-onglet "Onglets". Cliquez sur le bouton "Ajouter", et remplissez les champs avec le nom de la classe, "AdminTest". Ne confondez pas "class" et "modules" ! Choisissez l'icône (par exemple une provenant du pack FamFamFam), choisissez où l'onglet devra aller, et enregistrez. C'est fait ! Vous êtes libre maintenant d'optimiser cela.

En cas de problème

Si votre module ne marche pas comme prévu, voici quelques possibilités pour obtenir de l'aide :

Le forum officiel de PrestaShop

Rejoignez notre forum à l'adresse http://www.prestashop.com/forums/, et lancez une recherche sur les mots clés en rapport avec votre problème. Si la recherche a besoin d'être précisée, utilisez le formulaire de recherche avancée. Et si aucune recherche ne vous apporte de réponse utile, lancez une nouvelle discussion, où vous pourrez être aussi prolixe que nécessaire au moment de décrire votre situation ; il vous faut bien sûr être d'abord membre du forum.

Certains forums conservent des discussions en tête des autres discussions : ils contiennent des informations utiles, lisez-les bien.

Notre bug-tracker

S'il se trouve que votre problème vient d'un bug de PrestaShop plutôt que de votre code, envoyez un rapport de bug sur le bug-tracker de PrestaShop : http://forge.prestashop.com/ (il vous faudra vous enregistrer). Cela vous permet de discuter le problème directement avec les développeurs PrestaShop.

Sites officiels PrestaShop

Adresse	Description
http://www.prestashop.com	Site officiel de l'application PrestaShop, de sa communauté, et de la société qui l'édite.
http://addons.prestashop.com	Place de marché pour les thèmes et modules.
http://www.prestabox.com	Laissez-nous héberger votre boutique !