

Prologue

L'installation PrestaShop par défaut offre un thème neutre en noir et blanc, ce qui permet aux vendeurs de démarrer rapidement et librement leur activité, quel que soit leur métier.

Près de 2000 thèmes sont disponibles sur le marché PrestaShop Addons (<http://addons.prestashop.com/en/>). Ils ont été créés soit par des membres de la communauté PrestaShop, soit par PrestaTeam, et sont vendus à des prix raisonnables - certains sont même gratuits.

En tant que concepteur graphique / développeur web, vous pouvez également mettre vos thèmes à vendre sur le site PrestaShop Addons et gagner 70% du prix de vente.

Exigences

Votre code PHP devrait fonctionner sur PHP 5.4+.

Votre code HTML / CSS / JS devrait fonctionner avec au moins IE9 +, Edge, Firefox 45 et Chrome 29. Mobile-wise: iOS 8.4 et Android Browser 4.4

Norme et directives de codage

Guide général du code

Prévoyez des espaces pour chaque langue (PHP, HTML, CSS, etc.): 4 espaces pour les fichiers PHP, 2 espaces pour tous les autres types de fichiers.

Utilisez notre [fichier .editorconfig](https://github.com/PrestaShop/PrestaShop/blob/develop/.editorconfig) pour configurer facilement votre éditeur:
<https://github.com/PrestaShop/PrestaShop/blob/develop/.editorconfig>

Fichiers PHP

Vous devez suivre la [norme PSR-2](#) , tout comme PrestaShop le fait.

En général, nous avons tendance à respecter [les normes de codage de Symfony](#) .

Fichier HTML

Utilisez les balises HTML 5:

- `<br />` -> `<br>`,
- `<nav>`,
- `<section>`,
- etc.

Toutes les étiquettes ouvertes doivent être fermées dans le même fichier (un `<div>` **ne** doit **pas** être ouvert `header.tpl` puis fermé `footer.tpl`). Subtemplates (les modèles destinés à être inclus dans un autre modèle) doivent résider dans un `/_partials/` dossier.

CSS

Utilisez CSS3.

Nous vous recommandons de suivre la structure RSCSS: <http://rscss.io/>

JavaScript

Assurez-vous que votre outil Linter suit notre fichier .eslint: <https://github.com/PrestaShop/PrestaShop/blob/develop/.eslintrc>

Si vous souhaitez écrire le code ECMAScript 2015 (ES6), nous vous conseillons d'utiliser le compilateur Babel: <https://babeljs.io/>

Une bonne pratique JS consiste à diviser les fichiers par utilisation, puis à les compiler en un seul.

En savoir plus sur la norme ES2015: <https://babeljs.io/docs/learn-es2015/>

Migration depuis PrestaShop 1.6

PrestaShop 1.7 introduit un système de thème entièrement retravaillé à partir de la version 1.6, dans le but de faciliter les mises à jour de PrestaShop, et de faciliter ou de créer plus rapidement un nouveau thème.

L'énorme changement dans le système de thème signifie qu'il n'y a pas de moyen facile de migrer votre thème PrestaShop 1.6 vers PrestaShop 1.7.

Lorsque vous utilisez la mise à niveau automatique de PrestaShop 1.6 à 1.7, votre thème sera remplacé par le nouveau thème par défaut appelé "Classic". Nous vous conseillons donc de travailler sur votre thème 1.7 avant de passer au PrestaShop 1.7.

Certaines fonctionnalités ont été supprimées avec PrestaShop 1.7. Ils étaient déjà obsolètes en 1.6, ou se sont révélés trop problématiques pour maintenir.

Mise en route

Configuration de votre environnement local

Maintenant que vous avez l'intention de créer un thème pour PrestaShop, il est préférable de garder tout votre travail de développement sur votre machine. L'avantage principal est qu'il vous permet de contourner complètement la procédure de téléchargement de vos fichiers sur votre serveur en ligne afin de tester vos modifications. Un autre avantage est qu'un environnement de test local vous permet de tester le code sans risque de briser votre magasin de production. Le fait d'avoir un environnement local est la première étape essentielle du cheminement du développement web.

Remarque:

Le contenu suivant suppose que vous êtes un développeur et que vous souhaitez créer un thème ou un module.

Installation de PrestaShop

Nous vous conseillons d'installer PrestaShop en utilisant [Git](#) et [Composer](#).

Ensuite, ouvrez une ligne de commande sur votre répertoire de travail (vide), puis:

1. `git clone https://github.com/PrestaShop/PrestaShop.git`
2. `composer install`

En utilisant git, vous pouvez choisir votre version PrestaShop: . Nous vous préviendrons également de tester votre résultat final avec une version zip, juste pour la sécurité (puisque la version du fournisseur pourrait être légèrement différente).`git checkout 1.7.0.0`

Remarque:

Si vous ne l'avez pas encore fait, nous vous recommandons vivement de lire notre article [Configurer votre Git pour contribuer](#)

Création de votre fichier .gitignore

Un fichier gitignore est un incontournable pour tout projet Git-versionné, car il spécifie les fichiers intentionnellement non reliés que Git doit ignorer.

Quoi ignorer

Généralement, vous ne devez pas modifier les types de fichiers suivants:

- Fichiers temporaires (tels que les fichiers cache)
- Fichiers générés (tels que CSS minifié ou fichiers XML récupérés)
- Les fichiers avec des informations d'identification ou des informations personnelles (par exemple `settings.inc.php`)
- Fichiers OS et IDE (tels que `.DS_Store` ou `.idea /`)

```
actifs / css / *  
assets / js / *  
node_modules /
```

Nous vous suggérons de créer votre propre utilisation <http://gitignore.io>.

Pointe:

Si vous construisez un projet complet pour un client, vous pouvez lire notre [article sur la création d'un gitignore pour PrestaShop](#).

Créez votre thème depuis le thème Starter

Lorsque vous souhaitez créer un thème, la meilleure façon est d'utiliser StarterTheme comme thème de base.

Créez un nouveau dossier sous **themes**/le téléchargement de StarterTheme et copiez ses fichiers dans votre nouveau dossier.

[Télécharger StarterTheme](#)

Créez votre fichier theme.yml

Vous devez renommer tout d'abord, `config/theme.dist.yml` pour `config/theme.yml` et le modifier en fonction du nom de votre thème.

```
nom : YOUR_THEME_DIRECTORY_NAME
display_name : VOTRE THEME NOM
version : 1.0.0
auteur :
  nom : "PrestaShop Team" email : "pub@prestashop.com" url : "http://www.prestashop.com"

meta :
  compatibilité :
    de : 1.7.0.0
    à : ~
```

Une fois qu'il est terminé, vous pourrez sélectionner votre thème dans votre backoffice.

Thème de démarrage

PrestaShop 1.7 présente une nouvelle façon pour les concepteurs de créer leur thème à partir de zéro: le thème Starter. Le thème par défaut pour PS 1.7 est basé sur le thème Starter.

Pour pratiquement tous les CMS, le thème par défaut est utilisé comme cadre pour créer un thème personnalisé: les concepteurs doivent retravailler le thème par défaut et le remodeler dans ce qu'ils veulent afficher. Parfois, cela implique d'avoir à passer beaucoup de temps à supprimer toutes les règles CSS et au code JavaScript du thème par défaut, et à réécrire tout. Cela signifie beaucoup de travail avant même de commencer à créer quelque chose d'original.

Cela signifie que beaucoup de thèmes sont liés aux choix techniques du thème par défaut, car cette façon de travailler rend difficile de faire vos propres choix. Par exemple, comme le thème par défaut utilise Bootstrap, il est difficile d'utiliser Foundation.

Avec le thème Starter, l'équipe PrestaShop a décidé de créer un thème squelettique qui vous donnera une kickstart pour votre thème personnalisé, avec tout le code minimum (fichiers de modèle essentiels, markup et JavaScript) et suffisamment de liberté pour faire vos propres choix. Vous pouvez choisir d'utiliser Bootstrap, Foundation ou Blueprint. Le thème Starter n'est pas opinâtre: il n'y a pas de décision d'utiliser une bibliothèque ou une autre.

En utilisant le thème Starter comme base pour votre thème personnalisé, tout est prêt pour vous, il vous *suffit* de créer sur ce sujet.

Téléchargement du thème Starter

Le thème Starter est disponible sur GitHub: <https://github.com/PrestaShop/StarterTheme>

Si vous téléchargez StarterTheme et sélectionnez-le comme thème pour votre magasin, vous verrez un thème minimaliste avec un style trop simpliste (laid). Ce n'est qu'à des fins de développement. Vous ne devriez PAS utiliser le Starter Theme tel *quel*, et vous ne devriez PAS utiliser ses règles CSS par défaut ni les inclure dans votre thème: effacez tous les fichiers à l'intérieur de `_dev / css`.

CLASSIC

Normal

- [Home](#)
- [Grass](#)

- [Grass](#)

Grass

The Grass type (Japanese: くさタイプ Grass type) is one of the eighteen types. Notable Trainers who specialize in Grass-type Pokémon are the Gym Leaders Erika of Celadon City, Gardenia of Eterna City, Cilan, former Gym Leader of Striaton City, and Ramos of Coumarine City. Prior to changes in Generation IV, all damaging Grass-type moves were special, but they may now also be physical depending on the attack.

Products

Sort by: [Relevance](#) [Name, A to Z](#) [Name, Z to A](#) [Price, low to high](#) [Price, high to low](#)

 Bulbasaur Add to cart €284.40 €282.40 - Reduced price - New Available !	 Ivysaur Add to cart €339.90 €332.90 - On sale! - New Available !	 Venusaur Add to cart €492.00 €486.00 - On sale! - New Available !
		

Capture d'écran du StarterTheme avec style dev

Remarque:

La bibliothèque jQuery v2 est chargée par le fichier core.js.

Attention:

Veuillez noter que si vous souhaitez vendre votre thème sur le marché PrestaShop Addons, il existe des exigences spécifiques. Par exemple, les thèmes distribués par Addons DOIVENT utiliser Bootstrap 4.

Modifier. Ne pas annuler.

Lorsque vous souhaitez créer un nouveau thème, copiez et collez tous les fichiers du thème Starter dans votre répertoire de thème vide. Ensuite, vous commencez à le modifier et à créer votre propre thème.

Ne l'utilisez pas en tant que thème parent, vous ne rencontrerez que des problèmes et perdrez votre temps.

Une fois que vous avez supprimé tout style `_dev/css`, votre thème devrait être le suivant:

English

- [English](#)
- [Français](#)

[Sign in](#)

CLASSIC

- [Normal](#)

1. [Home](#)
2. [Grass](#)

- [Grass](#)
-

Grass

The Grass type (Japanese: くさタイプ Grass type) is one of the eighteen types. Notable Trainers who specialize in Grass-type Pokémon are the Gym Leader Leilan, former Gym Leader of Saffron City, and Ramos of Coumarine City. Prior to changes in Generation IV, all damaging Grass-type moves were special attacks.

Products

Sort by: [Relevance](#) [Name, A to Z](#) [Name, Z to A](#) [Price, low to high](#) [Price, high to low](#)



[Bulbasaur](#)

[Add to cart](#)

€284.40 €282.40

- Reduced price
- New

Available !



Capture d'écran du StarterTheme une fois que vous avez supprimé le style dev

Lire aussi: <http://build.prestashop.com/tag/starter-theme/>

Organisation thématique

Structure de l'annuaire

Un thème de PrestaShop est un ensemble de fichiers que vous pouvez modifier afin de changer l'apparence de votre boutique en ligne.

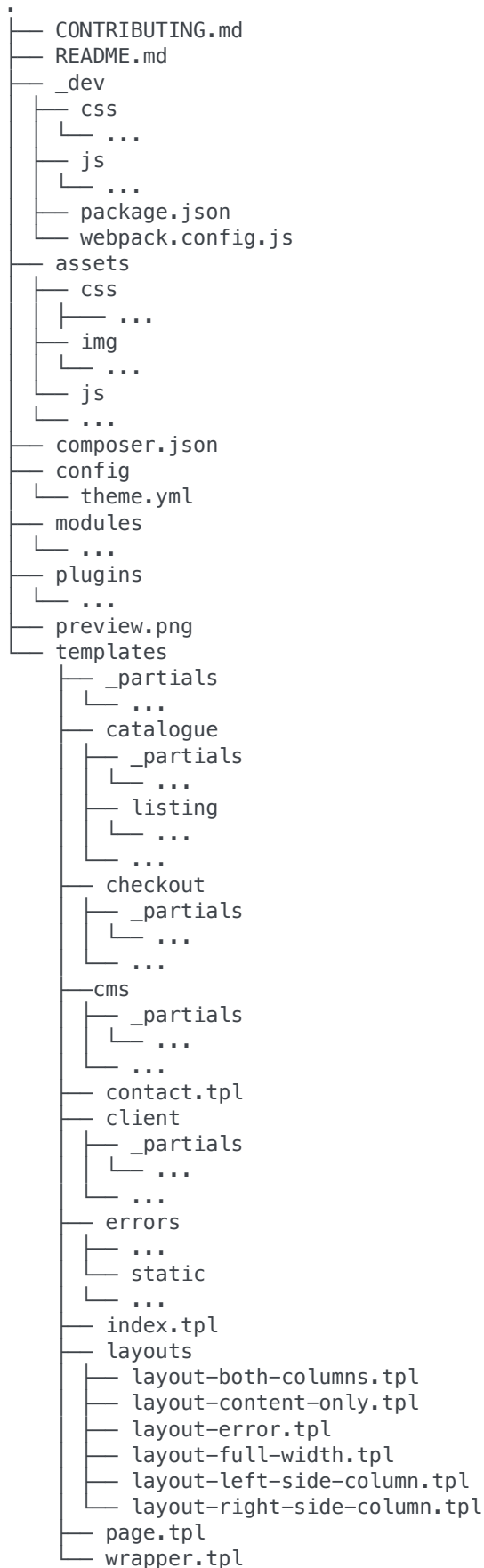
Voici quelques conseils importants:

- Tous les thèmes ont leurs fichiers situés dans le dossier / themes, à la racine du dossier de PrestaShop.
- Chaque thème a son propre sous-dossier, dans le dossier des thèmes principaux.

- Chaque thème est constitué de fichiers de modèle (.tpl), de fichiers d'image (.jpg, .png et autres), d'un ou plusieurs fichiers CSS (.css), et généralement des fichiers JavaScript (.js).
- Chaque thème dispose d'un fichier d'image preview.png dans son dossier, permettant au propriétaire du magasin de voir le thème qui ressemble directement au back office et de sélectionner le thème de manière appropriée.

La meilleure façon d'apprendre à créer un thème pour PrestaShop 1.7 est de plonger dans StarterTheme.

Voici son organisation, qui est expliquée plus en détail ci-dessous.



Les dossiers sont utilisés de cette façon:

Dossier	Objectif
<code>_dev</code>	Contient les fichiers de développement bruts pour vos ressources SCSS, JavaScript et image. Ils doivent être compilés à l'aide de Webpack et transformés en actifs de production.
les atouts	Contient les actifs de production, compilés par Webpack à partir des fichiers <code>_dev</code> .
config	Contient le fichier de configuration. Par défaut, il n'a que le fichier <code>theme.yml</code> .
module	Contient des modules spécifiques au thème ou la version du thème des fichiers de modèle des modules natifs. Par exemple, le <code>themes/classic/modules/ps_categorytree/views/templates/front/ps_categorytree.tpl</code> remplace le module de la catégorie <code>modules/ps_categorytree/views/templates/front/ps_categorytree.tpl</code>
plugins	Vos plugins personnalisés de smarty
les modèles	Contient les fichiers modèles eux-mêmes (.tpl), principalement dans des sous-dossiers contextuels (catalogue, caisse, cms, etc.). Le dossier <code>_partials</code> contient des "modèles partiels", ce qui signifie des éléments qui peuvent être utilisés par plusieurs modèles: <code>header.tpl</code> , <code>breadcrumb.tpl</code> , <code>footer.tpl</code> , etc. Cela empêche les blocs de code redondants et rend les thèmes plus faciles à entretenir.

Modèles et bibliothèques requis

Modèles requis

Lorsque vous installez / activez un thème, PrestaShop vérifie si le thème est valide: il recherche le fichier `theme.yml` (et vérifie son contenu), sa compatibilité déclarée et l'existence de certains fichiers.

Il existe une liste de fichiers qui doivent exister, même s'ils sont vides. Consultez la documentation dédiée pour savoir [ce qui rend un thème valide](#).

Il se pourrait que vous ayez construit une sorte de thème révolutionnaire et qu'il ne fonctionne pas comme le Starter Theme. Par exemple, si vous n'avez pas de page de produit, vous n'avez pas besoin du fichier `product.tpl`. Dans ce cas, il vous suffit de créer un fichier `product.tpl` vide. Soyez gentil avec le prochain développeur et ajoutez un commentaire indiquant où se trouve le code relatif aux produits;)

Bibliothèques requises

jQuery v2.1 est chargé par le `core.js` fichier central (inclus), mais pas d'autres bibliothèques, car l'idée est que le thème Starter ne devrait pas faire l'objet d'une opinion.

En savoir plus sur la [gestion des actifs](#).

Theme.yml

Le fichier `theme.yml` de la thème définit toute la configuration de la thème et les méta-informations, telles que son numéro de version, les mises en page, la plage de compatibilité, la configuration des crochets, etc.

Description du thème

Le nom du thème DOIT correspondre à son nom de répertoire. Par exemple, si le thème s'appelle "My Awesome Theme" et sa valeur 'name' est définie sur "my-awesome-theme", alors le dossier DOIT être / my-awesome-theme.

Les utilisateurs pourront choisir la mise en page pour chaque page de la page de paramètres du thème. Les mises en page sont automatiquement analysées dans le dossier du thème / modèles / mises en page, de sorte que cette clé de configuration est facultative, mais elle permet aux concepteurs de fournir des informations plus conviviales que simplement un nom de fichier.

```
Nom : StarterTheme # Le nom doit correspondre au nom du répertoire
display_name : Starter Theme
version : 1.0.0
```

```
theme_key : 3c6e0b8a9c15224a8228b9a98ca1531d # Nécessite par PrestaShop Addons
auteur :
  nom : "John Doe" email : "pub@prestashop.com" url : "http: //www.prestashop.com " meta : (
```

```
: Description de la mise en page pleine largeur
: Idéal pour les pages de produits pour maximiser la taille de l'image : colonne gauche
```

Paramètres globaux

Configuration

Vous pouvez faire modifier la configuration de PrestaShop lorsque le thème est activé.

```
global_settings :
  configuration :
    PS_QUICK_VIEW : false
    NEW_PRODUCTS_NBR : 4
    PS_PNG_QUALITY : 8
```

Modules

Vous pouvez utiliser les modules de validation ou de désactivation du thème lorsque le thème est activé.

```
global_settings :
  modules :
    to_enable :
      # Tous les modules ci-dessous sont activés lorsque
      # le thème est activé (et installé si nécessaire).
      # Ils sont désactivés lorsque le thème est désactivé.
      - my-custom-module
      - yippeeslider
    to_disable :
      # Tous les modules ci-dessous sont désactivés lorsque le thème est activé.
      # Ils sont réactivés lorsque le thème est désactivé.
      - homeslider
      - blockwishlist
```

Vous pouvez également créer la créativité de créer des crochets et joindre des modules aux hameçons personnalisés et existants lorsque le thème est activé.

```
global_settings :
  hooks :
    custom_hooks :
      - name : displayFooterBefore
        title : displayFooterBefore
        description : Ajoutez une zone de widget au-dessus du pied de page
    modules_to_hook :
      displayHeaderTop :
        # displayHeaderTop aura exactement les
        # modules suivants accrochés à elle dans l'ordre spécifié.
        # Chaque module de cette liste sera décoché
        # de tous les autres crochets d'affichage sur lesquels il est accroché.
        - blocklanguages
        - blockcurrencies
        - blockuserinfo
      displayHeaderMiddle :
```



```
# displayHeaderMiddle aura tout ce qui est actuellement accroché à
# # gardé accroché à elle, et blocksearch sera ajouté
# à la liste (ou déplacé à la fin si déjà accroché là).
- ~
- blocksearch
displayHeaderBottom :
# displayHeaderBottom aura blocktopmenu et blockcart
#
précédemment . - blocktopmenu
- blockcart
- ~
displayFooter :
- blocknewsletter
displayLeftColumn :
# blockcategories est accroché sur toutes les pages sur displayLeftColumn
- blockcategories
# blocktags est accroché sur displayLeftColumn sur toutes les pages
# sauf "catégorie" et "index"
- blocs :
    except_pages :
        - category
        - index
```

Paramètres d'image

L'activation du thème supprime tous les types d'image existants.

Par conséquent, les thèmes DOIVENT déclarer leurs types d'image, et ce qu'ils s'appliquent.

```
global_settings :
  image_types :
    cart_default :
      largeur : 80
      hauteur : 80
      portée : [ produits ]
    small_default :
      largeur : 125
      hauteur : 125
      portée : [ produits , catégories , fabricants , fournisseurs ]
    medium_default :
      largeur : 300
      hauteur : 300
      portée : [ produits , catégories : fabricants , fournisseurs ]
    large_default :
      largeur : 500
      hauteur : 500
      portée : [ produits ]
    home_default :
      largeur : 250
      hauteur : 250
      portée : [ produits ]
    category_default :
      largeur : 960
      hauteur : 350
      portée : [ catégories ]
    product_listing :
      largeur : 220
      hauteur : 220
      portée : [ produits , catégories , fabricants , fournisseurs ]
    large_banner :
      largeur : 960
      hauteur : 400
      portée : [ catégories ]
```

Paramètres de thème

Tous les paramètres ci-dessous peuvent être modifiés via une interface dans l'interface de back office du thème, et ne dépendent que de la combinaison thème / magasin.

Lorsque le fichier theme.yml est analysé par PrestaShop, la touche de configuration 'theme_settings' est copiée dans un fichier nommé settings_n.yml, où 'n' est l'identifiant du magasin où le thème est installé (settings_123456.yml, par exemple) .

Lorsque la configuration est modifiée via l'interface de back office, seul le fichier settings_n.yml est mis à jour - le fichier theme.yml reste inchangé.

```
global_settings :
  theme_settings :
    default_layout : layout-full-width
    layouts :
      # Disposition spécifique pour certaines pages
      identité : layout-side-side-column
      order-confirmation : layout-left-side-column
```

Dépendances

Lorsque vous créez un thème, vous pouvez ajouter des fonctionnalités à des modules personnalisés. Il est important que ces modules soient installés avec votre thème. Ces modules doivent être déclarés en tant que dépendances, de sorte que vous êtes sûr que prestashop les exportera lors de la création de votre thème avec zipball.

Jusqu'à présent, les thèmes ne comportent que des dépendances de modules.

```
dépendances :
  modules :
    - xx_customslider
    - xx_customproductpage
```

Aideurs: fonctions et modificateurs

{url}

PrestaShop 1.7 présente un nouveau Smarty helper pour générer des URL. Cela prend en charge SSL, le nom de domaine, URI de base physique et virtuelle, la concaténation de paramètres et, bien sûr, la réécriture d'URL.

{url} utilise la classe Link en interne.

Remarque:

Consultez l'`$url` ensemble de données pour trouver des URL déjà régénérées (comme la maison, le panier, la connexion, etc.).

Attention:

Une instance de l'objet Link est toujours transmise aux modèles à des fins de compatibilité ascendante, car elle a été fortement utilisée. **Il est fortement recommandé de ne pas l'utiliser**.

Voici quelques exemples:

```
{ url entity = adresse id = $ id_address }
{ url entity = adresse params = [ 'id_address' => $ id_address ] }
{ url entity = adresse id = $ id_address params = [ 'delete' => 1 ] }
```

... sortira respectivement:

```
http://prestashop.ps/it/address?id_address=3
http://prestashop.ps/it/address?id_address=3
http://prestashop.ps/it/address?id_address=3&delete=1
```

Widgets

{widget}

PrestaShop 1.7 présente une nouvelle façon d'afficher les modules dans un thème. Au lieu d'utiliser un crochet et d'accrocher votre module, la fonction du widget vous permet d'afficher tout contenu du module dans votre modèle.

Ceci est une caractéristique essentielle de PrestaShop 1.7, assurez-vous de [lire la documentation dédiée](#).

Voici un exemple du thème classique, il affiche les coordonnées du magasin où vous le souhaitez.

```
< div id = "sidebar" >
  { widget name = "ps_contactinfo" }
</ div >
```

Étant donné que le module peut avoir un comportement différent selon le crochet auquel ils sont accrochés, vous pouvez passer le nom du crochet.

```
< div id = "footer" >
  { nom du widget = "ps_contactinfo" hook = "displayFooter" } </ div >
```

{widget_block}

Encore mieux, vous pouvez réécrire le modèle sur la route. Le module utilisera votre code Smarty au lieu de charger le fichier modèle.

Prenant le [module ps_linklist](#) comme exemple. Au lieu d'utiliser `ps_linklist/views/templates/hook/linkblock.tpl`, vous pouvez l'annuler de cette façon:

```
{ Widget_block nom = "ps_linklist" }
{ foreach de linkBlocks de $ comme linkBlock $ }
< ul >
  { foreach linkBlock.links de $ comme $ link }
    < li >
      < h4 > < a href = " { link.url de $ } " > { $ link.title } </ a > </ h4 >
      < p > { $ link.description } </ p>
    </ li >
  { / foreach }
</ ul >
{ / foreach }
{ / widget_block }
```

{rendu}

Certains éléments provenant du contrôleur doivent être transmis à cette fonction. Jusqu'à présent, il n'est utilisé que pour les formulaires (informations client et caisse).

```
{ fichier de rendu = 'client / _partials / login-form.tpl' ui = $ login_form }
```

{form_field}

`{form_field}` la fonction vous aidera à créer des formulaires, elle peut être comparée aux assistants de back-office introduits dans PrestaShop 1.5. Cela vous aide à maintenir le marquage du formulaire très cohérent.

En tant que concepteur de modèles, vous trouverez le balisage de chaque élément `_partials/form-fields.tpl`.

```
{ champ form_field = $ field }
```

... où `$field` est un tableau, comme cet exemple:

```
$ field = [
  'name' => 'user_email',
  'type' => 'email',
  'required' => 1,
  'label' => 'Email',
  'value' => null,
  'availableValues' => [],
  'erreurs' => [],
];
```

Modificateurs de noms de classe

Pour utiliser les données du contrôleur pour générer des noms de classe, nous avons ajouté 2 modificateurs: 'classname' et 'classnames'.

classname

Le modificateur de données `classname` garantira que votre chaîne est un nom de classe valide.

Ce sera:

1. Mettez-le en minuscules.
2. Remplacez tous les caractères non ASCII (tels que les caractères accentués) par leur équivalent ASCII. [Voir le code ici](#) .
3. Remplacez tous les caractères non alphanumériques par un seul tiret.
4. Assurez-vous qu'un seul tiret consécutif est utilisé.

```
{ assign var = attr value = 'Hère-Is_a-Classname - @ # $$ * (& -----' }
{ $ attr | nom de classe }
```

... sortira:

```
here-is-a-classname-v
```

classnames

Ce modificateur de données prend un tableau, où la clé est le nom de classe et la valeur est un booléen indiquant s'il doit être publié ou non.

Notez que chaque nom de classe est passé par le `classname` modificateur.

```
$ body_classes = [
    "lang-fr" => true,
    "rtl" => false,
    "pays-FR" => true,
    "monnaie-EUR" => true,
    "schéma de pleine largeur" => true,
    " page-index "=> true,
];
```

De cette façon, ce code Smarty:

```
< Body class = " { $ page.body_classes | classnames } " >
```

... générera:

```
< body class = "lang-fr country-fr currency-eur layout-full-width page-index" >
```

Thème parent / enfant

PrestaShop 1.7 repose beaucoup sur l'héritage du modèle afin de créer le thème le plus cohérent possible tout en réduisant considérablement la quantité de code dupliqué.

Nous l'avons encore amené à présenter la fonctionnalité thème Parent Child. Le but est d'éviter de modifier le thème principal afin que vous puissiez le mettre à jour!

Remarque:

Cette fonctionnalité n'est utile que si vous souhaitez légèrement modifier un thème (pour ajouter un bloc par exemple). Si vous devez modifier le balisage plus profondément, modifiez le thème, ne créez pas de thème enfant.

Attention:

N'utilisez pas StarterTheme comme thème parent. StarterTheme doit être modifié.

Le principe

Jusqu'à présent, nous avons parlé de l'extension du modèle dans le même thème. Dans PrestaShop 1.7, vous pouvez maintenant étendre les modèles à partir d'un autre thème.

Pointe:

En tant que développeur de thème, vous souhaitez créer autant de blocs que possible pour que votre utilisateur puisse remplacer la quantité minimale de code.

Comment créer un thème enfant

D'abord, vous devez avoir le thème que vous souhaitez utiliser comme parent dans votre `/themes` dossier de magasin.

Ensuite, vous pouvez créer un thème très minimal, contenant uniquement les fichiers suivants:

```

├── config
│   └── theme.yml
└── preview.png
  
```

Pointe:

Il est recommandé de copier ces fichiers à partir du thème Parent.

Une fois que vous avez ceci, vous spécifiez dans votre thème enfant `theme.yml` quel thème doit être utilisé en tant que parent. La valeur doit être le nom technique du thème (c'est-à-dire le nom du dossier de la thème).

```

parent : nom classique
: childtheme display_name : Mon premier enfant Version Thème : 1.0.0 assets : use_parent_assets
  
```

Allez-y, sélectionnez ce thème dans votre back-office et vous êtes tous ensemble.

Modèles de substitution

Sans autre modification, le thème enfant utilisera tous les modèles du thème principal.

Dans l'exemple suivant, nous modifions le modèle de catégorie.

Redéfinir le modèle entier

Créez le modèle de catégorie dans votre thème enfant `templates/catalog/listing/category.tpl`. À ce stade, vous pouvez faire ce que vous voulez dans ce modèle, mais vous souhaitez probablement étendre le modèle de liste de produits. Dans l'affirmative, vous ne devez pas copier le modèle de liste de produits sur votre thème enfant, PrestaShop utilisera le fichier parent.

Prolongez la liste des produits de manière normale et annulez le bloc dont vous avez besoin.

```
{ extends file = 'catalogue / listing / product-list.tpl' }
```

Extension du même modèle

Une autre façon de remplacer le modèle de catégorie serait d'étendre le modèle parent et de définir uniquement le bloc dont vous avez besoin. Il y a une petite différence si vous souhaitez remplacer le même modèle, vous devez ajouter la ressource parent (notez le mot-clé parent dans l'exemple ci-dessous).

```
{ extends file = 'parent: catalogue / listing / category.tpl' }
```

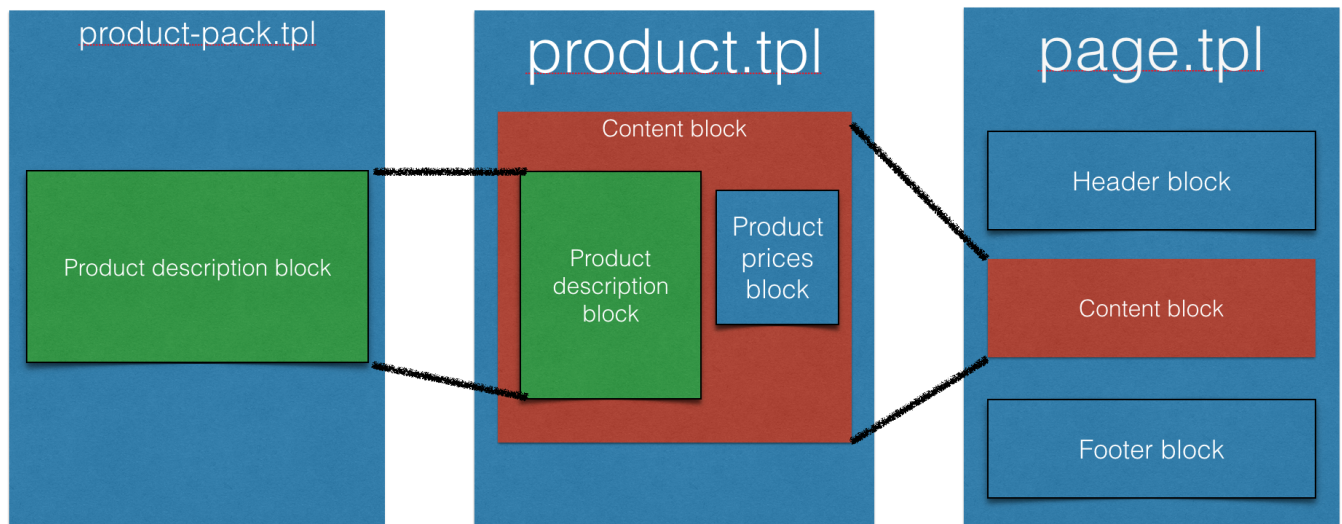
Modèle d'héritage

PrestaShop 1.7 repose beaucoup sur l'héritage du modèle afin de créer le thème le plus cohérent possible tout en réduisant considérablement la quantité de code dupliqué.

Le principe

L'héritage de modèle vous permet d'étendre un modèle parent et de redéfinir le bloc dont vous avez besoin.

La photo ci-dessous illustre l'exemple d'une page de produit spécifique étendue à une version générique. Dites que vous avez trois fichiers: `layout-pack.tpl` extension `product.tpl`, lui-même étendue `page.tpl`.



Le `product-page.tpl` fichier ne contiendra que la partie description du produit. Tout le reste sera exactement le même que la page produit. Encore mieux, la page du produit ne définira que le contenu principal de la page, tout le reste sera tiré de son propre modèle parent (c.-à-d. `Page.tpl`).

La mauvaise façon de faire cela serait d'extraire l'en-tête, le pied de page et la partie réutilisable du modèle et les inclure dans chaque fichier.

La [documentation officielle de Smarty](#) a un bon exemple. Dans leurs propres mots:

"L'héritage de modèle est une approche pour gérer des modèles qui ressemblent à des techniques de programmation orientées objet. Au lieu de l'utilisation traditionnelle de `{include ...}` tags pour gérer des parties de modèles, vous pouvez hériter les contenus d'un modèle à un autre (comme étendre une classe) et modifier des blocs de contenu (comme les méthodes de substitution d'une classe). la gestion des modèles est minimale et efficace, car chaque modèle ne contient que les différences par rapport au modèle qu'il étend."

Exemple de la vie réelle de PrestaShop

Dans un thème PrestaShop, de nombreuses pages sont très similaires, par exemple les produits de liste de modèles: catégories, nouveaux produits, résultats de recherche, etc. Tous affichent une liste de produits, donc dans PS 1.7 ils s'étendent tous `catalog/listing/product-list.tpl` (ce qui étend la disposition principale).

```
{ extends file = $ layout }

{ block name = 'content' }
< section id = "main" >

    { block name = 'product_list_header' }
    < h2 class = "h2" > { $ listing.label } </ h2 >
    { / block }

    { block name = 'product_list' }
    { include file = 'catalogue / _partials / products.tpl' listing = $ listing }
    { / block }
```

```

    </ section >
  { / block }

```

Le modèle affichera un titre et une liste de produits en dessous. Pour la page de catégorie, nous voulons une belle description avec une image de couverture. Nous pouvons donc simplement remplacer le *product_list_header*

```

{ extends file = 'catalogue / listing / product-list.tpl' }

{ nom du bloc = 'product_list_header' }

< h1 > { $ category.name } </ h1 >
< div class = "category-cover" >
  < img src = " { $ category.image.large.url } " alt = " { $ category.image.legend } " >
</ div >
< div id = " category-description " > { $ category.description nofilter } </ div >

{ / block }

```

Cela réduit le code au minimum, sans aucune répétition.

Asset Management

PrestaShop 1.7 a considérablement amélioré la façon dont les actifs (CSS, JavaScript et les fichiers image) sont gérés.

Nous conseillons aux développeurs de thème de compiler la plupart de leur style et de leur code JavaScript dans un seul fichier concaténé / minifié (voir la section Webpack ci-dessous).

Si vous devez ajouter des ressources spéciales, par exemple une bibliothèque JavaScript supplémentaire sur la page d'accueil ou la page du produit, il existe quelques façons de le faire.

Votre thème doit imprimer correctement les éléments dans le modèle smarty, et il est expliqué dans la section de modèle .

Enregistrement d'actifs

Dans PrestaShop 1.7+, il est facile d'enregistrer des actifs personnalisés sur chaque page. L'amélioration majeure est que vous pouvez facilement les gérer à partir de votre thème, sans aucun module.

Nous avons introduit de nouvelles méthodes pour enregistrer des actifs, et en particulier de nouvelles options intéressantes.

Par exemple, vous pouvez enregistrer votre actif spécifiquement dans la tête ou le bas de votre code HTML; vous pouvez le charger avec des attributs comme `async` ou `defer`; et vous pouvez même l'intégrer facilement.

Une option préférée est la *priorité*, ce qui rend très simple de s'assurer que tout est chargé dans l'ordre dont vous avez besoin.

Remarque:

La rétrocompatibilité est maintenue pour les `addJS()`, `addCSS()`, `addjQueryUI()` et `addjQueryPlugin()` méthodes. Incidemment, c'est le meilleur moment pour mettre à jour vos bibliothèques et utiliser la nouvelle méthode.

Voici une liste d'options et de ce qu'elles font.

Options

La classe `FrontController` de PrestaShop fournit 2 nouvelles méthodes pour enregistrer facilement de nouveaux éléments: `registerStylesheet()` et `registerJavascript()`.

Pour avoir les signatures les plus extensibles, ces 2 méthodes prennent 3 arguments. Le premier est l'ID unique de l'actif, le second est le chemin relatif, et le troisième est un ensemble de tous les autres paramètres optionnels, comme décrit ci-dessous.

ID

Cet identifiant unique est nécessaire pour chaque actif. Ceci est utile pour remplacer ou annuler l'enregistrement de quelque chose déjà chargé par le Core ou un module natif.

Chemin relatif

C'est le chemin de votre atout. Afin de rendre vos actifs entièrement débordables et compatibles avec la fonctionnalité parent / enfant, vous devez fournir le chemin du répertoire racine du thème, ou le répertoire racine de PrestaShop s'il s'agit d'un module.

Par exemple:

- 'assets / css / example.css' pour quelque chose sur votre thème.
- 'modules / modulename / css / example.cs' pour quelque chose dans votre module.

Paramètres supplémentaires pour la feuille de style

prénom	Valeurs	Commentaire
médias	tout braille embossed handheld imprimer projection écran dis- cours tty tv (par défaut: tous)	sans commentaires.

priorité	0-999 (par défaut: 50)	0 est la plus haute priorité
en ligne	true false (par défaut: false)	Si c'est vrai, votre style sera imprimé dans la <code><style></code> balise dans votre code HTML <code><head></code> . Utiliser avec précaution.

Paramètres supplémentaires pour JavaScript

prénom	Valeurs	Commentaire
position	tête bas (par défaut: en bas)	Les fichiers JavaScript doivent être chargés dans la partie inférieure autant que possible. Rappelez-vous: core.js est chargé d'abord en bas afin que jQuery ne soit pas chargé dans la partie <code><head></code> .
priorité	0-999 (par défaut: 50)	0 est la plus haute priorité
en ligne	true false (par défaut: false)	Si c'est vrai, votre style sera imprimé à l'intérieur des balises dans votre code HTML. Utiliser avec précaution. <code><script type="text/javascript"></code>
attribut	async defer none (par défaut: none)	Chargez le fichier JavaScript avec l'attribut correspondant (en savoir plus: Async vs Defer atributos)

Enregistré par Core

Chaque page de chaque thème charge les fichiers suivants:

- theme.css
- CSS personnalisé
- rtl.css (si une langue de droite à gauche est détectée)
- core.js
- theme.js
- custom.js

Nom de fichier	ID	Priorité	Commentaire
theme.css	thème principal	0	La plupart (tous?) Des styles de votre thème. Doit être minifié.
rtl.css	theme-rtl	900	Chargé uniquement pour la langue droite vers la gauche
CSS personnalisé	thème personnalisé	1000	Le fichier vide est chargé à la fin pour permettre à l'utilisateur de remplacer un style simple.
core.js	corejs	0	Fourni par PrestaShop. Contient JQuery2, dépêche les événements PrestaShop et détient la logique PrestaShop.
theme.js	thème principal	50	La plupart des JavaScript de votre thème. Devrait intégrer des bibliothèques requis sur toutes les pages, et être minifié.
custom.js	thème personnalisé	1000	Fichier vide chargé à la fin, pour permettre à l'utilisateur de remplacer le comportement ou d'ajouter un script simple.

Enregistrement en thèmes

À l'heure actuelle, vous avez probablement compris que ce `theme.yml` fichier est devenu le cœur des thèmes PrestaShop.

Pour enregistrer des actifs, créez une nouvelle `assets` clé au niveau supérieur de votre compte `theme.yml` et enregistrez vos fichiers en fonction de vos besoins. Les identifiants de page sont basés sur la `php_self` propriété de chaque contrôleur ([exemple](#))

Par exemple, si vous souhaitez ajouter une bibliothèque externe sur chaque page et une bibliothèque personnalisée sur la page Produit:

```
assets :
  css :
    product :
      - id : chemin du produit-extra-style
        : assets / css / product.css media : toute priorité : 100 js : all : - id : this-coo
```

```
200
attribut : asynchrone
```

Enregistrement en modules

Lors du développement d'un module PrestaShop, vous pouvez ajouter des styles spécifiques à vos modèles. La meilleure façon est d'utiliser les méthodes `registerStylesheet` et les `registerJavascript` méthodes fournies par la `FrontController` classe parent.

Remarque:

Si vous développez un module personnalisé qui ne fonctionne que sur vos thèmes, ne mettez pas de code de style ou de JavaScript dans le module: placez-le dans les fichiers du thème à la place (`theme.js` et `theme.css`).

Avec un module de contrôle avant

Si vous développez un contrôleur avant, il suffit d'étendre la `setMedia()` méthode. Par exemple:

```
Fonction publique setMedia ()
{
    parent :: setMedia ();

    if ('product' === $ this-> php_self) {
        $ this-> registerStylesheet (
            'module-modulename-style',
            'modules /'. $ this-> module-> nom. ' / css / modulename.css ',
            [
                ' media '=> ' all ',
                ' priority '=> 200,
            ]
        );

        $ this-> registerJavascript (
            'module-modulename-simple-lib',
            'modules /'. $ this-> module-> nom. ' / js / lib / simple-lib.js ',
            [
                ' priority '=> 200 ,
                'attribut' => 'async',
            ]
        );
    }
}
```

Sans module de contrôle avant

Si vous ne possédez que la classe de votre module, enregistrez votre code sur le hook `actionFrontControllerSetMedia` et ajoutez votre actif sur le chemin du hook:

```
Fonction publique hookActionFrontControllerSetMedia ($ params)
{
    // Seulement sur la page du produit
    si ('produit' === $ this-> context-> controller-> php_self) {
        $ this-> context-> controller-> registerStylesheet (
            'module- Modulename-style ',
            ' modules /'. $ this-> name. ' / css / modulename.css ',
```

```

        [
            'media' => 'all',
            'priority' => 200,
        ]
    );

    $ this-> context-> controller-> registerJavascript (
        'module-modulename-simple-lib',
        'modules /'.$ this-> name.' / js / lib / simple-lib.js ',
        [
            ' priority ' => 200,
            'attribut' => 'async',
        ]
    );
}

// Sur chaque page
$ this-> context-> controller-> registerJavascript (
    'google-analytics',
    'modules /'.$ this-> name.' / Ga.js ',
    [
        ' position '=>' head ' ,
        'inline' => true,
        'priority' => 10,
    ]
);
}

```

Désenregistrer

Vous pouvez annuler l'enregistrement des actifs! C'est tout le point d'un **id**. Par exemple, si vous souhaitez améliorer la compatibilité de votre thème / module avec un module, vous pouvez vous désinscrire de ses actifs et les gérer vous-même.

Disons que vous voulez être entièrement compatible avec un module de navigation populaire. Vous pouvez créer un remplacement de modèle, bien sûr, mais vous pouvez également supprimer le style fourni avec celui-ci et regrouper votre style spécifique dans votre **theme.css** (car il est chargé sur chaque page).

Pour annuler l'enregistrement d'un bien, vous devez connaître son identifiant.

En thèmes

À partir d'aujourd'hui, le seul moyen de désinscrire un actif sans module est de placer un fichier vide où le dépassement de module serait.

Si le module enregistre un fichier JavaScript placé **views/js/file.js**, il vous suffit de créer un fichier vide **modules/modulename/views/js/file.js**.

Cela fonctionne pour les actifs JavaScript et CSS.

Dans les modules

Les deux **unregisterJavascript** et les **unregisterStylesheet** méthodes ne prennent qu'un seul argument: l'ID unique de la ressource que vous souhaitez supprimer.

```

// Dans une
fonction publique du
contrôleur frontal setMedia () {
    parent :: setMedia ();

    $ this-> unregisterJavascript ('the-identifiant');
}

// Dans une
fonction publique de
classe de module hookActionFrontControllerSetMedia ($ params) {

```

```

$ this-> context-> controller-> unregisterJavascript ('the-identifiant');
}

```

À propos de Webpack

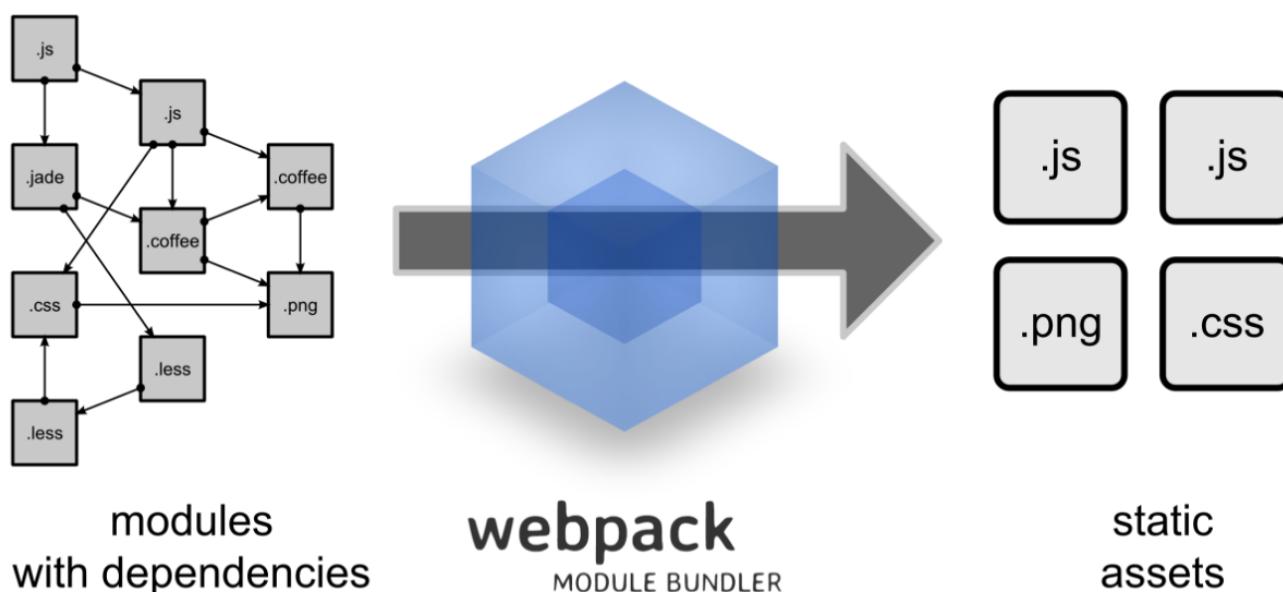
[Webpack](#) est un module bundler. Webpack prend des modules avec des dépendances et génère des éléments statiques représentant ces modules.

L'intérêt principal de l'utilisation de Webpack est qu'il compile tous vos styles - que nous vous conseillons d'écrire en utilisant [Sass](#) - dans un seul fichier CSS. De cette façon, votre thème ne fera qu'une seule requête HTTP pour ce fichier unique, et puisque votre navigateur le mettra en cache pour une réutilisation ultérieure, il ne téléchargera ce fichier qu'une seule fois.

Il en va de même pour votre code JavaScript. Au lieu de charger jQuery avec ses plugins communautaires, vos propres plugins personnalisés et tout code supplémentaire dont vous pourriez avoir besoin, Webpack compile et minifie tout ce code JavaScript dans un seul fichier, qui sera chargé une fois et en cache.

Remarque:

Webpack n'est pas du tout requis par PrestaShop, vous êtes libre d'utiliser votre outil préféré! La documentation explique Webpack car c'est l'outil que nous avons choisi pour le thème Classic, et StarterTheme est livré avec un fichier de configuration prêt à l'emploi.



Installation du webpack

Si vous souhaitez compiler vos actifs en utilisant Webpack (et nous vous le conseillons), procédez comme suit:

1. Téléchargez et installez [Node.js](#), qui contient l'outil npm.
2. Dans votre outil de ligne de commande, ouvrez le dossier `_dev`.
3. Installez npm: `npm installation`.
4. Vous avez alors le choix:
 - Pour créer vos actifs une fois, tapez `npm run build`.
 - Pour reconstruire vos actifs chaque fois que vous modifiez un fichier dans le dossier `_dev`, tapez `npm run watch`.

Configuration du Webpack

Le [fichier de configuration Webpack pour StarterTheme](#) est ainsi:

1. Toutes les règles CSS vont au fichier `assets / css / theme.css`.
2. Tout le code JavaScript passe dans le fichier `assets / js / theme.js`.

Il fournit une configuration appropriée pour compiler vos fichiers Sass, Less, Stylus ou CSS dans un seul fichier CSS.

Le code JavaScript est écrit en ES6 et compilé sur ES5 avec Babel.

Si vous souhaitez utiliser Stylus ou Less, modifiez simplement la ligne de commande sous la section "scripts".

```
var webpack = require ( 'webpack' );
var ExtractTextPlugin = require ( "extraire-text-webpack-plugin" );

var plugins = [];

plugins . push (
  nouveau ExtractTextPlugin ( '../css/theme.css' )
);

module . exports = [{
  //
  Entrée JavaScript : [
    './js/theme.js'
  ],
  sortie : {
    chemin : '../assets/js' ,
    nom de fichier : 'theme.js'
  },
  module : {
    loaders : [ {
      test : /\.js$/ ,
      exclure : / node_modules / ,
      chargeurs : [ 'babel-loader' ]
    }
  ],
  externals : {
    prestashop : 'prestashop'
  },
  plugins : plugins ,
  résoudre : {
    extensions : [ '' , '.js' ]
  }
}, {
  //
  Entrée CSS : [
    './css/normalize.css' ,
    './css/example.less' ,
    './css/st/dev.styl' ,
    './css/theme.scss'
  ],
  sortie : {
    chemin : '../assets/js' ,
    nom de fichier : 'theme.js'
  },
  module : {
    chargeurs : [{
      test : /\.scss$/ ,
      loader : ExtractTextPlugin . extrait (
        "style" ,
        "css-loader" sourceMap! postcss! sass-loader? sourceMap "
      )
    }, {
      test : /\.styl$/ ,
      loader : ExtractTextPlugin . extrait (
        "style" ,
        "css-loader" sourceMap! postcss! stylus-loader? sourceMap "
      )
    }, {
      test : /\.less$/ ,
      loader : ExtractTextPlugin . extrait (
        "style" ,
        "css-loader? sourceMap! postcss! less-loader? sourceMap"
      )
    }, {
      test : /\.css$/ ,
```

```

    loader : ExtractTextPlugin . extrait (
      'style' ,
      'css-loader' sourceMap! postcss-loader '
    )
  }, {
    test : /\.(png|woff(2)?|eot|ttf|svg)(\?[a-z0-9= \. ] +)? $ / ,
    loader : 'file-loader? name = .. / css / [hash]. [ext]'
  }]
},
plugins : plugins ,
résoudre : {
  extensions : [ '' , '.scss' , '.styl' , '.less' , '.css' ]
}
}];

```

Modèles et mises en page

Le fichier modèle PrestaShop est basé sur le [moteur de modèle Smarty 3](#).

Tous les fichiers modèles doivent être stockés dans le **templates/**sous - dossier du thème . Par exemple, le thème par défaut a ses fichiers de modèle dans le dossier suivant: **/themes/classic/templates**.

Structure de l'annuaire

Les modèles sont ensuite divisés entre différents sous-dossiers.

Dossier	La description
_partials	Code partagé sur l'ensemble du site comme l'en-tête, le pied de page ou les notifications.
catalogue	Product page, produit / marque / liste des fournisseurs, résultats de recherche et autres.
check-out	Panier, options de livraison, options de paiement, confirmations de commandes et autres.
cms	Tout le contenu statique: contact, sitemap, pages CMS et autres.
client	Tout sur le compte du client et ses données.
les erreurs	Tous les modèles d'erreur: introuvable, erreur du serveur, interdit et tel.
mises en page	Les modèles de thème: 1, 2 colonnes ou plus, largeur totale, tout est possible.

Les fichiers de modèle doivent être écrits afin qu'un seul .tpl puisse générer une page HTML entière - sauf s'il s'agit d'un **_partials** dossier ou d'un sous-dossier (voir notre norme de codage, liée à partir du chapitre Prologue de cette documentation).

Modèles

Nous faisons une **nette différence entre les modèles et la mise en page**.

- Un modèle étend une disposition.
- La disposition tient l'organisation globale de la page
- Un modèle spécifié pour une fonctionnalité: la page du produit par exemple

Il existe de nombreux modèles, un thème de PrestaShop, dont les principaux comprennent:

- **index.tpl** pour la page d'accueil
- **catalogue / product.tpl** pour la page produit
- **catalogue / listing / product-list.tpl** pour toute page de liste de produits
- **checkout / cart.tpl** pour le panier détaillé
- **checkout / checkout.tpl** pour le processus de caisse

Modèles spécifiques

Si vous travaillez sur un grand magasin dans de nombreuses langues, vous devrez modifier la disposition de la page en fonction de la langue.

Par exemple, vous voulez une page de produit différente pour les clients américains et japonais. Dans ce cas, il vous suffit de créer un nouveau modèle **product.tpl** et de le placer dans le bon dossier.

Lors de la recherche d'un modèle, PrestaShop vérifiera plusieurs sites pour déterminer quel fichier doit être utilisé. Il est très facile d'avoir un modèle différent pour une locale donnée ou un identifiant d'entité spécifique.

Avec la page du produit, le noyau vérifie les emplacements suivants (dans l'ordre) et renvoie le premier modèle trouvé:

Exemple avec un produit avec ID = 3 et locale = en-US

1. en-US/catalog/product-3.tpl
2. catalog/product-3.tpl
3. en-US/catalog/listing/product.tpl

4. catalog/listing/product.tpl

Un autre exemple avec le modèle de catégorie pour la catégorie avec ID = 9 et locale = en-US.

1. en-US/catalog/listing/category-9.tpl
2. catalog/listing/category-9.tpl
3. en-US/catalog/listing/category.tpl
4. catalog/listing/category.tpl
5. en-US/catalog/listing/product-list.tpl
6. catalog/listing/product-list.tpl

Cette fonctionnalité est principalement conçue pour les développeurs travaillant sur un modèle personnalisé pour un client.

Layouts

La mise en page est l'organisation de la page, la manière dont les parties de votre conception sont organisées. L'exemple typique est la barre latérale: existe-t-il une barre latérale sur votre page de catégorie ou si votre liste de produits prend l'espace entier?

Avec PrestaShop 1.7, les utilisateurs ont la possibilité de modifier le layout de chaque page de manière indépendante. En tant que développeur de modèles, votre rôle est de vous assurer que votre thème est compatible.

Page	Description	Layout
404 error	This page cannot be found	Full Width - No side columns, ideal for distraction-free pages such as product pages.
Best sales	Our best sales	Two Columns, small left column - Two columns with a small left column
Contact us	Use our form to contact us	Two Columns, small left column - Two columns with a small left column
Index	Shop powered by PrestaShop	☒ Full Width - No side columns, ideal for distraction-free pages such as product pages. ☐ Three Columns - One large central column and 2 side columns. ☐ Two Columns, small left column - Two columns with a small left column ☐ Two Columns, small right column - Two columns with a small right column
Manufacturers	Manufacturers list	Two Columns, small left column - Two columns with a small left column
New products	Our new products	Two Columns, small left column - Two columns with a small left column
Forgot your password	Enter the e-mail address you use to sign in to receive an e-mail with a new password	Full Width - No side columns, ideal for distraction-free pages such as product pages.
Prices drop	Our special products	Two Columns, small left column - Two columns with a small left column
Sitemap	Lost ? Find what your are looking for	Full Width - No side columns, ideal for distraction-free pages such as product pages.
Suppliers	Suppliers list	Full Width - No side columns, ideal for distraction-free pages such as product pages.

Contenu d'un fichier de mise en page

La mise en page est le niveau le plus élevé de : `<doc: `inheritance template <../smarty / template-inheritance.rst>`` tree. Fondamentalement, il tient les `<html>` étiquettes d'ouverture et de fermeture.

Les fichiers de mise en page typiques ressemblent à l'extrait suivant. Celui-ci est plein

Remarque:

N'oubliez pas de définir autant de blocs que possible.

```
<!doctype html>
<html lang = "{$ language.iso_code}">

<tête>
  {nom du bloc = 'tête'}
  {include file = '_partials / head.tpl'}
  {/bloc}
</ head>

<body id = "{$ page.page_name}" class = "{$ page.body_classes | classnames}">

  {hook h = 'displayAfterBodyOpeningTag'}

  <main>
```

```

<header id = "header">
  {nom du bloc = 'en-tête'}
  {include file = '_ partials / header.tpl'}
{/bloc}
</ header>

<section id = "wrapper">
  <div class = "container">

    {block name = 'breadcrumb'}
    {include file = '_ partials / breadcrumb.tpl'}
    {/bloc}

    {block name = "left_column"}
    <div id = "left-column">
      {if $ page.page_name == 'produit'}
        {hook h = 'displayLeftColumnProduct'}
      {autre}
        {hook h = "displayLeftColumn"}
      {/si}
    </ div>
    {/bloc}

    {block name = "content_wrapper"}
    <div id = "content-wrapper">
      {block name = "content"}
        <p> Bonjour tout le monde! C'est HTML5 Boilerplate. </ P>
      {/bloc}
    </ div>
    {/bloc}

  </ div>
</ section>

<footer id = "footer">
  {nom du bloc = "pied de page"}
  {include file = "_ partials / footer.tpl"}
{/bloc}
</ footer>

</ main>

{hook h = 'displayBeforeBodyClosingTag'}

{block name = 'javascript_bottom'}
  {include file = "_ partials / javascript.tpl" javascript = $ javascript.bottom}
{/bloc}

</ body>

</ html>

```

De là, chaque partie du thème fera son travail et remplacera le contenu à l'intérieur de ces briks, en gardant la même organisation.

Chef

La partie principale est très importante en termes de référencement et de performances.

Regardez StarterTheme et Classic head part pour voir des exemples concrets.

Actifs

PrestaShop 1.7 a changé la façon dont fonctionne l'actif et cela signifie que la façon d'ajouter à l'intérieur de la `<head>` balise de votre page a également changé.

Il existe 2 fichiers importants à utiliser:

- `_partials/stylesheets.tpl`
- `_partials/javascript.tpl`

Il existe 2 fichiers utilisés pour profiter pleinement des nouvelles fonctionnalités de 1.7, comme le chargement asynchrone pour javascript ou automatique en ligne pour CSS.

Si vous utilisez StarterTheme (ce que je crois que vous faites), il n'y a rien à faire, cela fonctionne hors de la boîte.

Attention:

Il `_partials/javascript.tpl` doit également être inclus au bas de votre page.

```
{block name = 'stylesheets'}
  {include file = "_partials / stylesheets.tpl" stylesheets = $ stylesheets}
{/block}

{nom du bloc = 'javascript_head'}
  {include file = "_partials / javascript.tpl" javascript = $ javascript.head vars = $ js_c
{/block}
```

Ces 2 sous-couches sont très simples, elles bouclent et impriment chaque élément fourni.

SEO

Beaucoup d'informations de méta et de références appartiennent ici, il y a un bloc spécial pour tel que le modèle qui étend cette disposition peut redéfinir facilement le titre ou la description de la page.

```
{ nom du bloc = 'head_seo' }

<title>{block name='head_seo_title'}{$page.meta.title}{/block}</title>
<meta name="description" content="{block name='head_seo_description'}{$page.meta.descripti
<meta name="keywords" content="{block name='head_seo_keywords'}{$page.meta.keywords}{/blo

{if $page.meta.robots != 'index'}
  <meta name="robots" content="{ $page.meta.robots }">
{/if}

{if $page.canonical}
  <link rel="canonical" href="{ $page.canonical }">
{/if}

{ / block }
```

Notifications

Tout au long de l'ensemble du front-office, le client peut recevoir des messages de notification de PrestaShop pour l'informer sur les réussites ou les erreurs par exemple. Votre thème peut également envoyer des notifications lorsque certains événements se produisent.

Les messages de notification ne sont pas codés en dur dans les fichiers de modèle, mais ils sont envoyés par le contrôleur, de sorte que vous ayez une cohérence dans le cas où vous mettez à jour / modifiez votre thème. En outre, de cette façon, il y a de meilleures chances que tous les messages de notification soient déjà traduits dans votre langue!

Types de notifications

Un tableau de notification est transmis aux modèles, contenant au moins un de ces éléments:

Succès	Une action a été effectuée et tout s'est bien passé.
Erreur	Quelque chose a mal tourné.
Attention	Avis important que le commerçant devrait connaître.
Info	"Juste pour que tu saches".

Comment afficher les notifications

Dans le thème Starter, les [notifications sont mises en œuvre en tant que fichier modèle partiel](#) :

```
<aside id = "notifications">
  <div class = "container">
    { if $ notifications.error }
      <article class = "alert alert-danger" role = "alert">
        <ul>
          { foreach $ notifications.error comme $ notif }
            <li> { $ notif } </ li>
          { / foreach }
        </ ul>
      </ article>
    { / if }

    { si $ notifications.warning }
      <article class = "alert alert-warning" role = "alert">
        <ul>
          { foreach $ notifications.warning comme $ notif }
            <li> { $ notif } </ li>
          { / foreach }
        </ ul>
      </ article>
    { / if }

    { if $ notifications.success }
      <article class = "alert alert-success" role = "alert">
        <ul>
          { foreach $ notifications.success comme $ notif }
            <li> { $ notif } </ li>
          { / foreach }
        </ ul>
      </ article>
    { / if }

    { si $ notifications.info }
      <article class = "alert alert-info" role = "alert">
        <ul>
          { foreach $ notifications.info comme $ notif }
            <li> { $ notif } </ li>
          { / foreach }
        </ ul>
      </ article>
    { / if }
```

```
</ div>  
</ aside>
```

... et sont ensuite inclus dans le fichier modèle :

```
{ nom du bloc = 'notifications' }  
{ include file = '_partials / notifications.tpl' }  
{ / block }
```

Ajoutez votre propre message dans votre contrôleur avant

Votre contrôleur frontal conserve les 4 variables suivantes :

- `$this->error`
- `$this->success`
- `$this->warning`
- `$this->danger`

Ce sont des tableaux PHP, et ils contiennent des messages en tant que chaîne.

Depuis PrestaShop 1.7, vous pouvez [rediriger le client et afficher un message après une action](#) .

```
$ this-> success [] = $ this-> l ('Informations mises à jour avec succès.');
```

```
$ this-> redirectWithNotifications ($ this-> getCurrentURL ());
```

Pages de liste

Votre catalogue est principalement 2 choses: une liste de produit et une page de produit détaillée.

Cette section couvre les pages de liste, qui comprennent: la catégorie, le résultat de la recherche, les produits par marque, la liste des best-sellers, la nouvelle liste de produits, etc.

Afin de réduire la duplication de code, le seul modèle nécessaire est le fichier `catalog/listing/product-list.tpl`.

Élargir le modèle de liste de produits

Nous avons déjà couvert : [doc: 'comment prestashop choisit le bon modèle pour utiliser <010-templates-layouts.rst>'](#) afin que nous sachions que le modèle de catégorie étend le modèle de liste de produits.

Nous avons déjà couvert la façon dont l'intégrité du modèle vous permet de redéfinir une petite partie du modèle plus grand .

Donc, fondamentalement, vous êtes tous prêts à créer un modèle de catégorie ou un modèle de résultat de recherche qui fait beaucoup plus que le modèle de liste de produits!

Mise à jour de la page AJAX

Votre liste de produits changera à mesure que le client filtre le résultat avec des options de navigation ou de tri sélectif, par exemple.

L'une des règles d'or de StarterTheme et PrestaShop 1.7 est: **Aucune duplication de code de présentation** . Par conséquent, nous ne voulions pas retourner les données de json sur le résultat et laisser javascript reconsidérer la page.

Nous avons créé le noyau pour générer le sous-modèle et le renvoyer au client. En fin de compte, javascript est uniquement utilisé à l'endroit du contenu des espaces réservés HTML.

Chaque appel ajax régénèvera les modèles suivants:

- `catalog/_partials/products-top.tpl`
- `catalog/_partials/products.tpl`
- `catalog/_partials/products-bottom.tpl`

Comment mettre à jour la vue

PrestaShop émettra des événements JavaScript pour vous informer de quoi faire.

Exemple:

```
importer $ de 'jquery' ;
import de prestashop de 'prestashop' ;
importer 'velocity-animate' ;

$ ( document ). ready ( () => {
  prestashop . on ( 'updateProductList' , ( data ) => {
    updateProductListDOM ( données );
  });
}

function updateProductListDOM ( données ) {
  $ ( '#search_filters' ). replaceWith ( data . rendered_facets );
  $ ( '# js-active-search-filters' ). replaceWith ( data . rendered_active_filters );
  $ ( '# js-product-list-top' ). replaceWith ( data . rendered_products_top );
  $ ( '# js-product-list' ). replaceWith ( data . produits rendus );
  $ ( '# js-product-list-bottom' ). replaceWith ( data . rendered_products_bottom );
}
```

Modules

Modules de substitution

Lorsque vous écrivez un thème, vous devez souvent remplacer les modèles et les éléments provenant d'un module afin qu'ils correspondent aux besoins de balisage spécifiques de votre thème. Les thèmes peuvent remplacer les actifs des modules (CSS et JavaScript uniquement) en plaçant le fichier correspondant à un emplacement spécifique.

Avec PrestaShop 1.7, tout le code de remplacement du module va vers le répertoire des *modules* (dans le répertoire de votre module). Chaque développeur du module PS 1.7 doit être conscient de cette modification (introduit avec PR 5020: <https://github.com/PrestaShop/PrestaShop/pull/5020>).

Remarque:

Si votre annulation d'actif est vide, PrestaShop ne charge rien (ni le module un ni le remplacement). Ceci est utile si vous souhaitez supprimer complètement ce style de module et ajouter le vôtre à votre compilation **theme.css**.

Les exemples de cette page sont basés sur cette structure de répertoire de module exemple:

```

├──
├── css
│   ├── external-lib.css
│   └── style.css
├── js
│   └── app.js
├── moduledemo.php
├── views
│   └── templates
│       ├── front
│       │   ├── included-template.tpl
│       │   └── moduledemo.tpl

```

5 répertoires, 6 fichiers

Modification des modèles et des actifs

Avec PrestaShop 1.7, voici les chemins de dossier à créer afin de remplacer les modèles et les actifs:

```

├──
├── modules
│   ├── css
│   │   ├── external-lib.css
│   │   └── style.css
│   ├── js
│   │   └── app.js
│   └── views
│       └── templates
│           ├── front
│           │   ├── included-template.tpl
│           │   └── moduledemo.tpl

```

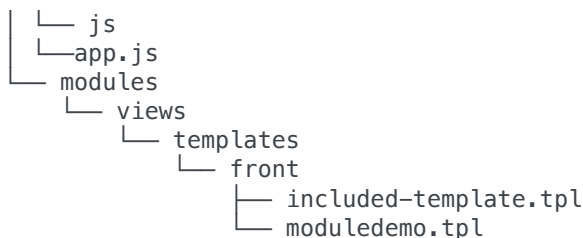
6 répertoires, 5 fichiers

Par rapport à ce qui était nécessaire dans PrestaShop 1.6, il est beaucoup plus simple:

```

├──
├── css
│   ├── modules
│   ├── css
│   ├── external-lib.css
│   └── style.css
├── js
│   └── modules

```



10 répertoires, 5 fichiers

Remplacer par la méthode 'include'

Il y a une question très importante que vous devriez connaître. Lors du chargement d'un fichier modèle (par exemple 'moduledemo.tpl'), PrestaShop recherchera d'abord le premier ordre, dans l'ordre suivant:

1. /themes/THEME_NAME/modules/MODULE_NAME/views/templates/front/moduledemo.tpl
2. /modules/MODULE_NAME/views/templates/front/moduledemo.tpl

Mais si votre fichier *moduledemo.tpl* inclut le fichier *inclus-template.tpl*, vous devrez également remplacer 'included-template.tpl', même si vous ne souhaitez pas le modifier (ni modifier le chemin). Cela signifie que tous les fichiers dont un fichier surchargé inclut doivent être copiés en tant que tel est pour que votre remplacement fonctionne correctement.

La question revient dans les deux sens: si vous souhaitez modifier le fichier *inclus-template.tpl*, vous devez remplacer le fichier *moduledemo.tpl* qui l'inclut.

```
{ include file = './included-template.tpl' }
```

PrestaShop 1.7 présente un nouveau moyen générique d'inclure des fichiers dans des modèles de modules. En utilisant cette méthode, toutes les règles attendues seront suivies:

```
{ include file = 'module: MODULE_NAME / views / templates / front / included-template.tpl' }
```

SmartyDev vous aide à déboguer!

PrestaShop 1.7 présente également notre propre outil SmartyDev, une extension Smarty qui vous permet de voir le nom du modèle dans votre balisage HTML géré. Cela aidera à débiter beaucoup, surtout en raison de la substitution de modèle.

Voici un exemple de marquage généré avec SmartyDev activé:

```
[...]
    < a href = "http://prestashop.ps/en/" class = "menu déroulant items" > English </
    </ li >
    </ ul >
</ div >
<!-- fin /Users/julien/Sites/PrestaShop/themes/classic/modules/blocklanguages/blocklanguages.

<!-- begin /Users/julien/Sites/PrestaShop/themes/classic/modules/blockuserinfo/blockuserinfo.
< div class = "user-info" >
    < i class = "icônes matérielles _gray-dark" > & # xE7FF; </ I >
    < a class = "Déconnexion" href = "http://prestashop.ps/fr/?mylogout=" rel = "nofollow"
    < = "http://prestashop.ps/fr/mon-compte" title = "Voir mon compte client" rel = "nofollow"
    </ div >
<!-- fin /Users/julien/Sites/PrestaShop/themes/classic/modules/blockuserinfo/blockuserinfo.

<!-- commencer /Users/julien/Sites/PrestaShop/themes/classic/modules/blockcart/blockcart.tpl
< div class = "blockcart cart-preview" data-refresh-url = "http://prestashop.ps/fr/"
    < div class = "header" >
        < a rel = "nofollow" href = "" # titre = "panier" >

[...]
```

Pour l'utiliser, il suffit de configurer la constante `_PS_MODE_DEV_` sur `true` dans le fichier `/config/defines.inc.php` de votre installation : ajoutez `define ('_PS_MODE_DEV_', true);` alignez-vous sur ce fichier pour activer le mode PrestaShop Developer, qui

comporte SmartyDev.

JavaScript

PrestaShop 1.7 a retraité beaucoup de code javascript, presque réécrit tout.

Il est recommandé d'en savoir plus sur ([Gestion des actifs PrestaShop](#)) avant de continuer.

Un magasin par défaut charge beaucoup moins de fichiers en 1.7 par rapport à 1.6, par exemple, il n'y a pas de fichiers spécifiques par page. Les 2 nouveaux fichiers importants que vous devez maîtriser sont:

Fichier	Contenu
core.js	Chargez JQuery2, effectuez des appels ajax, définissez la méthode principale que tous les utilisateurs devraient utiliser
theme.js	Associe tous les codes spécifiques aux thèmes et les bibliothèques

En savoir plus ([sur leur priorité](#)).

Événements

Déposer un événement

La meilleure façon de déclencher un événement est d'utiliser l' `prestashopobj`. Voici un exemple simple:

```

Prestashop . Émettez (
  'produit mis à jour' ,
  {
    dataForm : someSelector . serializeArray (),
    productOption : 3
  }
);

```

Événements distribués

PrestaShop émettra de nombreux événements `core.js` afin que votre code puisse s'en prévaloir

Nom de l'événement	La description
Mise à jour	Sur la page du panier, chaque fois que quelque chose se produit (changer la quantité, retirer le produit, etc.), le panier est rechargé par un appel ajax. Une fois le panier mis à jour, cet événement est déclenché.
Mise à jourAddressForm	Dans le formulaire d'adresse, certaines entrées déclencheront des appels ajax pour modifier le formulaire (comme le changement de pays), une fois le formulaire mis à jour, cet événement est déclenché.
UpdateDeliveryForm	Pendant la caisse, si l'adresse de livraison est modifiée, cet événement sera déclenché.
modifiéCheckoutStep	Chaque soumission aux étapes de paiement lancera cet événement.
updateProductList	Sur chaque page de liste de produits (catégorie, résultats de recherche, prix, etc.), la liste est mise à jour via les appels ajax si vous modifiez les filtres ou les options de tri. Chaque fois que le DOM est rechargé avec une nouvelle liste de produits, cet événement est déclenché.
clickQuickView	Si votre thème le gère, cet événement sera déclenché lors de l'utilisation, cliquez sur le lien de la vue rapide.
updateProduct	Sur la page du produit, la sélection d'une nouvelle combinaison va recharger le DOM via les appels ajax. Après la mise à jour, cet événement est déclenché.
Plus d'événements?	Contribuer au document pour décrire plus d'événements!

Déclenchement des événements délégués

Nous utilisons la délégation d'événement pour nous assurer que les événements sont toujours joints après la modification du DOM (comme après un appel ajax).

Voici un moyen simple de déclencher un événement délégué.

```
var corps = $ ( 'corps' ); // Nos événements sont habituellement attachés au corps

var event = jQuery . Événement ( 'clic' );
événement . cible = corps . trouver ( '.js-theClassYouNeed' );

corps . déclencheur ( événement );
```

FO Hooks

Cette section de la documentation concerne uniquement les crochets de la façade: affichage et action.

Tous les crochets

// TODO crée une page avec un tableau répertoriant le contenu du fichier *hooks.json* .

Création d'un crochet personnalisé

Création d'un crochet dynamique

Lorsque votre module ou votre thème appelle un crochet, PrestaShop l'exécute.

À partir d'un fichier PHP régulier:

```
Hook :: exec ( 'MyCustomHook' );
```

À partir d'un modèle Smarty:

```
{ hook h = 'MyCustomHook' }
```

Mettre votre crochet visible et réutilisable

Si vous souhaitez que l'utilisateur puisse voir votre crochet dans la page Position de PrestaShop (dans le back office), il doit être enregistré.

Vous pouvez enregistrer votre crochet à partir du fichier *theme.yml* de votre thème :

[SEE] theme-yml

```
global_settings :
  hooks :
    custom_hooks :
      - name : displayFooterBefore
        title : displayFooterBefore
        description : Ajoutez une zone de widget au-dessus du pied de page
```

Vous pouvez également enregistrer votre crochet à partir d'un module:

Distribution

Maintenant que vous avez créé un thème incroyable, vous voudrez probablement le lancer. La documentation suivante vous guidera en créant un zip et en passant la validation des add-ons.

Mais d'abord, vous devez tester votre thème!

Test

Qu'est-ce qui rend un thème valide

Lorsque vous essayez de sélectionner un thème dans le backoffice, PrestaShop vérifiera si votre thème est valide. Il ne sera pas installé si le thème n'est pas valide.

Un thème est valide s'il contient certains fichiers et certaines clés de configuration.

Fichiers obligatoires

Voici la liste complète des fichiers requis:

- preview.png
- config / theme.yml
- assets / js / theme.js
- assets / css / theme.css
- templates / _partials / form-field.tpl
- templates / catalog / product.tpl
- templates / catalogue / listing / product-list.tpl
- templates / checkout / cart.tpl
- templates / checkout / checkout.tpl
- templates / cms / category.tpl
- templates / cms / page.tpl
- templates / customer / address.tpl
- templates / customer / addresses.tpl
- templates / client / guest-tracking.tpl
- templates / client / guest-login.tpl
- templates / customer / history.tpl
- templates / customer / identity.tpl
- templates / index.tpl
- templates / customer / my-account.tpl
- templates / checkout / order-confirmation.tpl
- templates / customer / order-detail.tpl
- templates / customer / order-follow.tpl
- templates / customer / order-return.tpl
- templates / customer / order-slip.tpl
- templates / errors / 404.tpl
- templates / errors / forbidden.tpl
- templates / checkout / cart-empty.tpl
- templates / cms / sitemap.tpl
- templates / cms / stores.tpl
- templates / customer / authentication.tpl
- templates / customer / registration.tpl
- templates / contact.tpl

Touches de configuration requises

Votre fichier de configuration `config/theme.yml`(en [savoir plus sur ce fichier](#)) doit préciser certaines clés de configuration.

Voici la liste des clés que PrestaShop recherchera. Le point représente l'imbrication.

- prénom
- Afficher un nom
- version
- nom de l'auteur
- meta.compatibility.from
- meta.available_layouts
- global_settings.image_types.cart_default
- global_settings.image_types.small_default
- global_settings.image_types.medium_default
- global_settings.image_types.large_default
- global_settings.image_types.home_default
- global_settings.image_types.category_default
- theme_settings.default_layout

Comment utiliser PrestaShop Automated Test Suite

PrestaShop 1.7 est livré avec une suite de test automatisée utilisée pour développer Classic. Vous devez les utiliser pour vous assurer que votre thème est entièrement compatible avec la fonctionnalité de PrestaShop.

Vous avez besoin `node.js`, `npm`, `java` et Chrome installé sur votre machine.

Attention:

La suite de tests est **destructrice**. Ne l'exécutez pas en production.

Préparez votre magasin pour les tests:

1. `cd tests/Selenium`
2. `npm install`
3. Installez PrestaShop, de préférence avec 2 langues (bien que les tests soient idéalement les langages et les paramètres agnostiques)
4. Copier `tests/Selenium/settings.dist.js` à `tests/Selenium/settings.js` et personnaliser en fonction de votre configuration
5. Une fois que PrestaShop est installé, exécutez . **AVERTISSEMENT: ne jamais le faire dans un magasin de production car il modifiera les produits existants sans demander votre permission.** `prepare-shop.php`

La suite de tests utilise Selenium,

`webdriver.io` vous permet d'effectuer presque toutes les actions qu'un navigateur ferait en utilisant une API basée sur les promesses courantes. Vous aurez besoin d'une certaine connaissance des promesses de tirer le meilleur parti de l'outil.

Ajouter un nouveau test

Si vous souhaitez créer un test personnalisé, vous devriez connaître certaines choses. Les tests sont contenus dans le `specs` sous-dossier.

Jusqu'à ce que nous puissions faire plus de documentation, regardez les tests existants et l' [API WebDriver.io](https://webdriver.io).

Si vous devez ajouter des fonctions d'aide générale pour vos tests, ils devraient entrer `commands/init.js`.

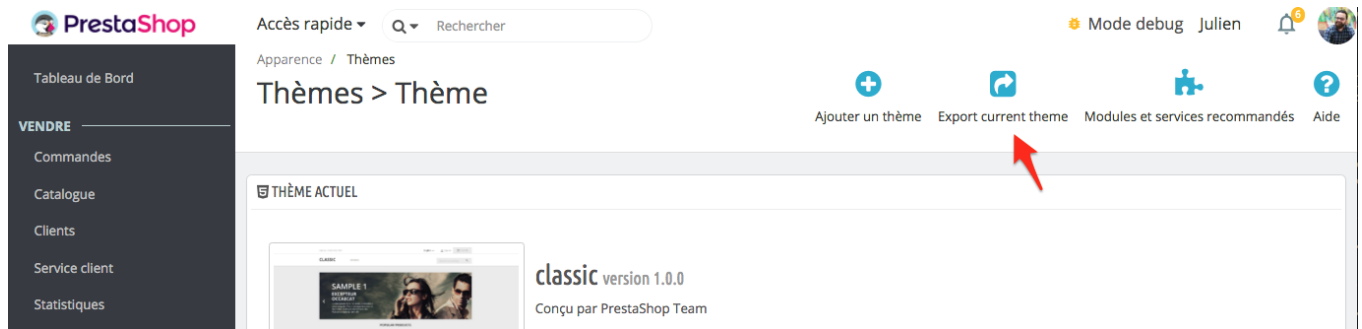
Si vous avez besoin d'appareils pour vos tests, utilisez ceux de l'installation par défaut ou fournissez un script qui les installe.

Ne codez pas les choses telles que les identifiants de produits dans vos tests: résumez-les derrière un nom et placez-les dans le `fixtures.js` fichier.

Création d'un fichier zip valide

Il n'y a plus de données de thème dans la base de données avec PrestaShop 1.7. Par conséquent, une matière est installée dès qu'il est sur le disque.

Si vous souhaitez que le thème apparaisse dans le backoffice, il suffit simplement de contenir un `config/theme.yml` fichier. Cela ne l'affichera que si vous souhaitez le sélectionner comme thème actif, il doit être valide. Lire [Ce qui fait un thème valide](#)



Une fois qu'il est actif, vous pouvez exporter votre thème à l'aide du bouton _ "Exporter le thème actuel" _ ou utiliser la commande à partir de votre terminal.

Ce qui est exporté

L'exportation de votre thème à l'aide du bouton ou de la ligne de commande exportera les données suivantes:

- Tous les fichiers de thème dans le répertoire
- Dépendances spécifiées dans `theme.yml` ([Voir theme.yml doc](#))
- Traductions par thème

Distribution sur les addons

Notez que si vous souhaitez vendre votre thème sur les Addons de PrestaShop, il existe quelques règles à suivre:

** LISTE DES EXIGENCES **

- Utilisez Bootstrap 4 alpha 4 - [suivez le document approprié](#) .
- Ajoutez votre clé - [suivez le document approprié](#) .

Principes fondamentaux du développement de PrestaShop

PrestaShop a été conçu afin que les modules tiers puissent facilement s'appuyer sur ses fondations, ce qui en fait un logiciel de commerce électronique extrêmement personnalisable.

La personnalisation de PrestaShop repose sur trois possibilités:

- Thèmes,
- Modules,
- Primordial.

Les thèmes sont explorés en entier dans le Guide de conception: TODO: Lien vers le Guide du concepteur. Les modules et le système de suppression sont explorés dans ce Guide du développeur, en commençant par la section «Concepts» ci-dessous. Vous pouvez en apprendre davantage sur chacun des chapitres suivants:

- TODO: lien vers la page "Création d'un module PrestaShop"
- TODO: Lien vers la page "Remplacer les comportements par défaut"

Remarque:

Par défaut, PrestaShop dispose de plus de 100 modules, vous permettant de lancer votre entreprise en ligne rapidement et gratuitement.

Plus de 2300 modules sont également disponibles sur le site officiel des add-ons. Ces modules supplémentaires ont été construits par la société PrestaShop ou les membres de la communauté PrestaShop et sont vendus à des prix abordables. En tant que développeur, vous pouvez également partager vos modules sur ce site et recevoir 70% des montants associés à la vente de vos créations. S'inscrire maintenant!

Concepts

Vous devez être familiarisé avec PHP et la programmation orientée objet avant d'essayer d'écrire votre propre module.

Vous pouvez apprendre PHP ici:

- <http://www.php.net/manual/en/getting-started.php>
- <http://www.codecademy.com/tracks/php>

Vous pouvez apprendre la programmation orientée objet ici:

- http://fr.wikipedia.org/wiki/Object-oriented_programming
- <http://net.tutsplus.com/tutorials/php/object-oriented-php-for-beginners/>

Un module est une extension de PrestaShop qui permet à tout développeur d'ajouter ce qui suit:

- Fournir des fonctionnalités supplémentaires à PrestaShop.
- Voir les éléments supplémentaires sur le site (sélection du produit, etc.).
- Communiquer avec d'autres services de commerce électronique (guides d'achat, plateformes de paiement, logistique, etc.).

La substitution est un système en soi. PrestaShop utilise un code complètement orienté objet. L'un des avantages est que, avec la bonne architecture de code, vous pouvez facilement remplacer ou étendre des parties du code de base avec votre propre code personnalisé, sans avoir à toucher le code de base. Votre code surpasse ainsi le code principal, ce qui permet à PrestaShop de se comporter comme vous le préférez. Il n'est pas recommandé d'utiliser une substitution dans un module que vous souhaitez distribuer (par exemple via le marché PrestaShop Addons), et ils sont interdits dans les modules partenaires. Gardez-les pour votre propre magasin.

L'architecture technique de PrestaShop

PrestaShop est basé sur une architecture à 3 niveaux:

- Objet / données. L'accès à la base de données est contrôlé par des fichiers dans le dossier "classes".
- Contrôle des données. Le contenu fourni par l'utilisateur est contrôlé par des fichiers dans le dossier racine.
- Conception. Tous les fichiers du thème sont dans le dossier "thèmes".

C'est le même principe que l'architecture Model View Controller (MVC), uniquement de manière plus simple et plus accessible.

Bien que toutes les versions de PrestaShop jusqu'à 1,6 fussent fières de l'utilisation d'une architecture personnalisée, il a été décidé d'intégrer le cadre Symfony PHP à partir de PrestaShop 1.7.

L'idée de conduite est que nous voulons que notre code soit plus robuste, plus modulaire et entièrement vérifiable. L'architecture 1.6, héritée de la version 1.5 et des années de développement de PrestaShop, n'est pas plus jeune et son âge commence vraiment à apparaître.

L'utilisation d'un framework open source éprouvé et populaire nous permettra de nous concentrer sur notre code métier principal (gérer un panier, gérer les commandes, calculer les prix et les taxes, générer des factures, etc.) avec une plus grande efficacité tout en profitant de la stabilité d'un système reconnu mondialement cadre.

Une architecture à 3 niveaux présente de nombreux avantages:

- Il est plus facile de lire le code du logiciel.
- Les développeurs peuvent ajouter et éditer un code plus rapidement.
- Le graphiste et les intégrateurs HTML peuvent travailler avec les limites du dossier / themes sans avoir à comprendre ni à lire une seule ligne de code PHP.
- Les développeurs peuvent travailler sur des données et des modules supplémentaires auxquels les intégrateurs HTML peuvent utiliser.

Modèle

Un modèle représente le comportement de l'application: traitement de données, interaction de base de données, etc.

Il décrit ou contient les données qui ont été traitées par la demande. Il gère ces données et garantit son intégrité.

Voir

Une vue est l'interface avec laquelle l'utilisateur interagit.

Son premier rôle est d'afficher les données fournies par le modèle. Son deuxième rôle est de gérer toutes les actions de l'utilisateur (clic de souris, sélection d'élément, boutons, etc.) et envoyer ces événements au contrôleur.

La vue ne fait aucun traitement; il affiche uniquement le résultat du traitement effectué par le modèle et interagit avec l'utilisateur.

Contrôleur

Le contrôleur gère les événements de synchronisation entre le modèle et la vue, et les actualise selon les besoins. Il reçoit tous les événements de l'utilisateur et déclenche les actions à effectuer.

Si une action a besoin de données à modifier, le Contrôleur "demandera" au Modèle de modifier les données et, à son tour, le Modèle notifiera à la Vue que les données ont été modifiées afin que la Vue puisse se mettre à jour.

Schéma de base de données

Vous pouvez télécharger le schéma SQL PrestaShop 1.6 en format PDF (3.90 Mb) ou dans le format de fichier MySQL Workbench (vous aurez besoin de MySQL Workbench pour le visualiser).

Configuration de votre environnement de développement local

Maintenant que vous avez l'intention de développer pour PrestaShop, il est préférable de garder tout votre travail de développement sur votre machine. L'avantage principal est qu'il vous permet de contourner complètement la procédure de téléchargement de votre fichier sur votre serveur en ligne afin de le tester. Un autre avantage est qu'un environnement de test local vous permet de tester le code sans risque de briser votre magasin de production. Avoir un environnement local est la première étape essentielle dans la voie du développement web.

Installation de PrestaShop localement

Prérequis

PrestaShop a besoin de la configuration suivante du serveur pour s'exécuter:

- Système: Unix, Linux ou Windows.
- Service Web: Apache Web Server 1.3 ou toute version ultérieure.
- PHP: 5.4 ou version ultérieure.
- MySQL: 5.0 ou version ultérieure.
- RAM du serveur: plus le meilleur ...

PrestaShop peut également fonctionner avec le serveur Web IIS de Microsoft 6.0 ou ultérieur, et nginx 1.0 ou version ultérieure.

Installation d'un environnement local

L'installation de n'importe quelle application Web local exige que vous installiez d'abord l'environnement adéquat, à savoir le serveur web Apache, l'interprète de langage PHP, le serveur de base de données MySQL et idéalement l'outil phpMyAdmin. C'est ce qu'on appelle un paquet AMP: Apache + MySQL + PHP et le système d'exploitation, donnant WAMP (Windows + Apache + MySQL + PHP), MAMP (Mac OS X + ...) et LAMP (Linux + ...). Étant donné que tous les éléments emballés sont open-source, ces installateurs sont la plupart du temps gratuits.

Voici une sélection d'installateur AMP gratuit:

- XAMPP: <http://www.apachefriends.org/fr/xampp.html> (Windows, Mac OS X, Linux, Solaris)
- WampServer: <http://www.wampserver.com/en/> (Windows)
- EasyPHP: <http://www.easyphp.org/> (Windows)
- MAMP: <http://www.mamp.info/> (Mac OS X)

Configuration de PHP

PrestaShop a besoin de quelques ajouts à PHP et MySQL afin de travailler pleinement. Assurez-vous que votre configuration PHP possède les paramètres et les outils suivants:

- Bibliothèque GD. La bibliothèque GD (<https://libgd.github.io/pages/about.html>) permet à PrestaShop de retravailler les images que vous téléchargez, notamment en les redimensionnant.
- Dom extension. L'extension Dom permet d'analyser les documents XML. PrestaShop l'utilise pour diverses fonctionnalités, comme le Localisateur de magasin. Il est également utilisé par certains modules, ainsi que par la bibliothèque `pear_xml_parse`.
- `allow_url_fopen` activée. La `allow_url_fopen` directive permet aux modules d'accéder aux fichiers distants, ce qui constitue une partie essentielle du processus de paiement, entre autres choses. Il est donc impératif de l'avoir mis sur ON.

Voici une section du `php.ini` fichier (le fichier de configuration pour PHP):

```
extension = php_mysql.dll
extension = php_gd2.dll
allow_url_fopen = On
```

a également recommandé

```
register_globals = Off
magic_quotes_gpc = Off
allow_url_include = Off
```

Téléchargement et extraction des fichiers PrestaShop

Vous pouvez télécharger la dernière version de PrestaShop à l'adresse <http://www.prestashop.com/fr/downloads>.

Vous pouvez télécharger la version de développement (instable) sur Github:

<https://github.com/PrestaShop/PrestaShop/archive/development.zip>

Extrayez les fichiers PrestaShop et placez-les dans le dossier racine du programme d'installation AMP que vous avez choisi:

- XAMPP: C:\xampp\htdocs ou / Applications / xampp / htdocs
- WampServer: C:\wamp\www
- EasyPHP: C:\easyphp\www
- MAMP: / Applications / MAMP / htdocs /

Création d'une base de données pour votre magasin local

Ouvrez l'outil phpMyAdmin à l'aide de votre navigateur. Son emplacement dépend du paquet AMP que vous avez choisi:

- <http://127.0.0.1/phpmyadmin> (XAMPP, WampServer, MAMP),
- <http://127.0.0.1/mysql> (EasyPHP)

Dans l'onglet "Bases de données", indiquez le nom de la base de données que vous souhaitez et validez en cliquant sur le bouton "Créer une base de données".

Installation de PrestaShop

Ouvrez le programme d'installation de PrestaShop, qui doit être situé à <http://127.0.0.1/prestashop/install>, et suivez ses instructions.

Vous pouvez lire le Guide de mise en route pour plus de détails: <http://doc.prestashop.com/display/PS16/Getting+Started>.

Configuration de PrestaShop

Par défaut, PrestaShop est configuré pour fournir un environnement sécurisé et stable à l'administrateur du magasin et aux clients.

En tant que développeur, il existe plusieurs modifications que vous pourriez et devriez apporter à l'installation par défaut afin de vous aider à coder mieux, localiser les bugs plus rapidement et généralement créer un excellent produit PrestaShop.

Désactivation du cache et contrainte de la compilation Smarty

Lorsque votre développement a un impact sur le front-office, que vous construisiez un thème ou simplement un module qui affiche des informations au client, vous devez forcer la compilation du fichier modèle et désactiver le cache afin de toujours voir le résultat de vos modifications directement.

Accédez à la page "Performances" dans le menu "Paramètres avancés" pour modifier les paramètres Smarty suivants:

- Cache de modèles: passez à "Force compilation".
- Cache: désactivez-le.

Le forçage de la compilation de Smarty ralentira toujours le temps de chargement de la page. Assurez-vous que votre magasin de production est configuré pour recompiler uniquement les modèles s'il existe des fichiers mis à jour et que son cache est activé.

Afficher les messages d'erreur

Les paramètres par défaut de PrestaShop empêchent le client de voir un message d'erreur du serveur ou un code de débogage.

Vous, d'autre part, avez besoin de cette information afin de corriger toute erreur potentielle dans votre code. À cette fin, ouvrez le `/config/defines.inc.php` fichier et modifiez-le pour configurer `_PS_MODE_DEV_` sur `true`:

```
/* Débogage uniquement */
define ( '_PS_MODE_DEV_' , true );
```

Utilisation de la méthode dump ()

Ces deux méthodes vous permettent de vérifier l'état d'une variable donnée à un endroit spécifique de votre code.

PrestaShop a utilisé des méthodes de débogage personnalisées pour les développeurs: `p($variable)` et `d($variable)`. Ils ont été utilisés pour afficher le contenu d'une variable et étaient vraiment une enveloppe autour de la `print_r()` méthode bien connue. En plus de cela, PrestaShop avait les méthodes `ppp()` et `ddd()`, qui étaient respectivement les alias de `p()` et `d()`; et les diverses `fd()`, `epr()` et `dieObject()` méthodes.

Tout cela a changé avec PrestaShop 1.7: nous comptons uniquement sur la fonction `VarDumper::dump()` de `Symfony`. Vous pouvez maintenant compter sur une seule méthode au lieu d'avoir plusieurs fonctions de débogage. La `dump()` fonction est très puissante et est toujours disponible en sortie (même sans `die()`). En outre, vous pouvez l'utiliser dans vos scripts de console car cette fonction dispose également d'un formatateur en mode CLI.

La `dump()` méthode n'est pas activée par défaut. Pour l'activer, vous devez activer le mode Debug, en définissant `_PS_MODE_DEV_` sur `true` (voir ci-dessus).

Activation du mode multistore

Depuis la version 1.5, PrestaShop est capable d'héberger plus d'un magasin dans une seule installation du logiciel. De nombreux administrateurs de magasins choisissent d'activer cette fonctionnalité, et cela peut avoir un impact significatif sur la manière dont fonctionne PrestaShop. Vous devriez donc vous assurer que tout ce que vous codez pour PrestaShop fonctionne en mode simple et multi-magasins.

L'activation du mode multi-magasins est simple: accédez à la page des préférences générales et mettez l'option "Activer Multistore" sur "Oui".

Vous pouvez basculer entre le mode magasin et le mode multi-magasins? En mode magasin unique, seul le magasin principal est utilisé.

Vous pouvez en savoir plus sur le mode multistore dans le Guide de l'utilisateur PrestaShop 1.6: <http://doc.prestashop.com/display/PS16/Managing+Multiple+Shops>.

À propos des fichiers de configuration

Il existe trois fichiers de configuration principaux, tous situés dans le `/config` dossier:

- `config.inc.php`
- `defines.inc.php`
- `smarty.config.inc.php`

config.inc.php

C'est le fichier de configuration principal de PrestaShop. Vous ne devriez pas toucher quoi que ce soit là-dedans.

define.inc.php

Ce fichier contient des valeurs constantes PrestaShop.

Il contient également l'emplacement de tous les fichiers et dossiers. Si vous devez modifier leur emplacement, n'oubliez pas de garder le chemin d'accès original à proximité, par exemple dans un commentaire PHP, au cas où vous devriez revenir plus tard.

En mode développement / test, vous devez vous assurer que tous les messages d'erreur s'affichent:

- Réglez sur `.define('_PS_MODE_DEV_', false);true`

Au contraire, en mode de production, vous devez cacher autant que possible les messages d'erreur!

- Assurez-vous que cette option est activée `.define('_PS_MODE_DEV_', false);false`

smarty.config.inc.php

Ce fichier contient tous les paramètres liés à Smarty.

Le système de mémoire cache Smarty doit toujours être désactivé, car il n'est pas compatible avec PrestaShop: conservez tel quel `$smarty->caching = false;`

`$smarty->compile_check`devrait être laissé `false` en mode développement.

`$smarty->debugging` donne accès aux informations de débogage Smarty lors de l'affichage d'une page. Ce paramètre est plus facilement modifié dans la page "Performance" du menu des paramètres avancés: l'option "Debug console" vous permet de choisir entre ne jamais afficher les informations de débogage de Smarty, toujours l'afficher ou l'afficher uniquement lorsque vous ajoutez `?SMARTY_DEBUG` à l'URL de la page que vous souhaitez tester, ce qui peut être très utile.

En mode de production, `$smarty->force_compile` doit être mis à faux, car cela donnera une augmentation de 30% du temps de chargement de votre page.

D'autre part, lors de l'édition d'un `.tpl` fichier, vous devez supprimer le `/tools/smarty/compiled` dossier (sauf le) afin de voir vos modifications appliquées. `index.php file`

Notez que ce paramètre peut être effectué directement à partir du back office, dans la page "Performance" sous le menu "Paramètres avancés".

Garder les choses sûres

Une fois que votre module est en ligne, vous pouvez accéder à ses fichiers par Internet. Même s'ils ne peuvent déclencher que des erreurs PHP, vous voudrez peut-être éviter que cela ne se produise.

Vous pouvez y accéder en ajoutant un fichier 'index.php' à la racine de n'importe quel dossier de module que vous créez. Voici une suggestion pour savoir quoi mettre dans le fichier.

```
En-tête ( "Expire: lun. 26 juil 1997 05:00:00 GMT" );
en-tête ( "Dernière modification:" . gmdate ( "D, d MYH: i: s" ) . "GMT" );
en-tête ( "Cache-Control: no-store, no-cache, must-revalidate" );
en-tête ( "Cache-Control: post-check = 0, pré-check = 0" , false );
en-tête ( "Pragma: no-cache" );
En-tête ( "Emplacement: ... /" );
sortie ;
```

Accès à la base de données

La structure de base de données

Par défaut, les tables de base de données de PrestaShop commencent par le préfixe `ps_`. Cela peut être personnalisé lors de l'installation

Tous les noms de table sont en minuscules et les mots sont séparés par un caractère de soulignement ("_"):

- `ps_employee`
- `ps_manufacturer`
- `ps_product`
- `ps_product_comment`
- `ps_shop_url`

Lorsqu'un tableau établit les liens entre deux entités, les noms des deux entités sont mentionnés au nom de la table. Par exemple, `ps_category_product` lie les produits à leur catégorie.

Quelques détails à noter sur les tableaux:

- Les tableaux contenant des traductions doivent se terminer par le suffixe `_lang`. Par exemple, `ps_product_lang` contient toutes les traductions pour la table `ps_product`.
- Les tableaux contenant les enregistrements reliant un magasin spécifique doivent se terminer par le suffixe `_shop`. Par exemple, `ps_category_shop` contient la position de chaque catégorie en fonction du magasin.

Il existe également quelques pratiques standard pour les lignes de données dans une table:

- Utilisez le champ `id_lang` pour mémoriser la langue associée à un enregistrement.
- Utilisez le champ `id_shop` pour stocker le magasin associé à un enregistrement.

La classe DBQuery

La classe `DBQuery` est un générateur de requêtes qui vous aide à créer des requêtes SQL. Par exemple:

```
$ sql = new DbQuery ();
$ sql-> select ('*');
$ sql-> de ('cms', 'c');
$ sql-> innerJoin ('cms_lang', 'l', 'c.id_cms = l.id_cms ET l.id_lang ='. (int) $ id_lang);
$ sql-> où ('c.active = 1');
$ sql-> orderBy ('position');
retour Db :: getInstance () -> executeS ($ sql);
```

Voici quelques-unes des méthodes de cette classe:

Nom et paramètres de la méthode	La description
<code>__toString ()</code>	Générer et obtenir la requête.
<code>construire()</code>	Générer et obtenir la requête (renvoyez une chaîne).
<code>de (chaîne \$ table, mixed \$ alias = null)</code>	Définir la table pour la clause FROM.
<code>groupBy (chaîne \$ champs)</code>	Ajoutez une restriction GROUP BY.
<code>ayant (string \$ restriction)</code>	Ajoutez une restriction dans la clause HAVING (chaque restriction sera séparée par une instruction AND).
<code>innerJoin (string \$ table, string \$ alias = null, string \$ on = null)</code>	Ajoutez une clause INNER JOIN, Eg \$ this-> innerJo in ('product p ON ...').
<code>rejoindre (string \$ join)</code>	Ajoutez une clause JOIN, Eg \$ this-> join ('RIGHT JOIN'. <i>DB_P</i> REFIX.'product p ON ...') ;
<code>leftJoin (chaîne \$ table, string \$ alias = null, string \$ on = null)</code>	Ajouter une clause LEFT JOIN.

Nom et paramètres de la méthode	La description
leftOuterJoin (chaîne \$ table, string \$ alias = null, string \$ on = null)	Ajoutez une clause LEFT OUTER JOIN.
limite (string \$ limit, mixed \$ offset = 0)	Limiter les résultats dans la requête.
naturalJoin (chaîne \$ table, string \$ alias = null)	Ajouter une clause NATURAL JOIN.
orderBy (string \$ fields)	Ajouter une restriction ORDER B.
sélectionnez (chaîne \$ champs)	Ajouter des champs dans la sélection de requête.
où (string \$ restriction)	Ajouter une restriction dans la clause WHERE (chaque restriction sera séparée par une instruction AND).

La classe ObjectModel

Lorsque vous avez besoin de plonger en profondeur, vous devez utiliser la classe ObjectModel. C'est l'objet principal du modèle d'objet de PrestaShop. Il peut être annulé ... avec précaution.

C'est un type de classe Active Record (voir: http://en.wikipedia.org/wiki/Active_record_pattern). Les attributs de table ou les attributs de vue de la base de données PrestaShop sont encapsulés dans cette classe. Par conséquent, la classe est liée à un enregistrement de base de données. Une fois que l'objet a été instancié, un nouvel enregistrement est ajouté à la base de données. Chaque objet récupère ses données à partir de la base de données; Lorsqu'un objet est mis à jour, l'enregistrement auquel il est lié est également mis à jour. La classe implémente les accesseurs pour chaque attribut.

Définition du modèle

Vous devez utiliser la variable statique \$ définition afin de définir le modèle.

Par exemple:

```

/ **
* Exemple à partir du modèle CMS (CMSCore)
* /
public static $ definition = array (
    'table' => 'cms',
    'primaire' => 'id_cms',
    'multilang' => true,
    'fields' => array (
        'id_cms_category' => array ('type' => self :: TYPE_INT, 'validate' => 'isUnsignedInt'),
        'position' => tableau ('type' => self :: TYPE_INT),
        'active' => array ('type' => self :: TYPE_BOOL),

        // Champs de langue
        'meta_description' =>
            array ('type' => self :: TYPE_STRING, 'lang' => true, 'validate' => 'isGenericName',
        'meta_keywords' =>
            array ('type' => self :: TYPE_STRING, 'lang' => true, 'validate' => 'isGenericName',
        'meta_title' =>
            array ('type' => self :: TYPE_STRING, 'lang' => true, 'validate' => 'isGenericName',
        'link_rewrite' =>
            array ('type' => self :: TYPE_STRING, 'lang' => true, 'validate' => 'isLinkRewrite',
        'contenu' =>
            array ('type' => self :: TYPE_HTML, 'lang' => true, 'valide' => 'isString', 'size' =>
    ),
);

```

Un modèle pour de nombreux magasins et / ou langues

Pour récupérer un objet dans plusieurs langues:

```
'multilang' => true
```

Pour récupérer un objet en fonction du magasin actuel

```
'multishop' => true
```

Pour récupérer un objet qui dépend du magasin actuel et dans plusieurs langues:

```
'multilang_shop' => true
```

Les principales méthodes

Toute modification des méthodes ObjectModel est susceptible d'influencer la façon dont toutes les autres classes et méthodes agissent. Utilisez avec précaution.

Nom et paramètres de la méthode	La description
__construct (\$ id = NULL, \$ id_lang = NULL)	Construire l'objet.
Ajouter (\$ autodate = true, \$ nullValues = false)	Enregistrez l'objet actuel dans la base de données (ajouter ou mettre à jour).
associerTo (nombre entier	array \$ id_shops)
effacer()	Supprimez l'objet actuel de la base de données.
deleteImage (mixed \$ force_delete = false)	Supprimez les images associées à l'objet.
deleteSelection (\$ selection)	Supprimez plusieurs objets de la base de données.
getFields ()	Préparez les champs pour la classe ObjectModel (ajouter, mettre à jour).
getValidationRules (\$ className = CLASS)	Règles de validation des objets de retour (validité du champ).
save (\$ nullValues = false, \$ autodate = true)	Enregistrez l'objet actuel dans la base de données (ajouter ou mettre à jour).
toggleStatus ()	Basculer l'état de l'objet dans la base de données.
mise à jour (\$ nullValues = false)	Mettre à jour l'objet actuel vers la base de données.
validateFields (\$ die = true, \$ errorReturn = false)	Vérifiez la validité du champ avant l'interaction de la base de données.

The Dispatcher

Le Dispatcher gère les redirections d'URL.

Par exemple, au lieu d'utiliser plusieurs fichiers dans le dossier racine comme `product.php`, `order.php` ou `category.php`, PrestaShop utilise un seul fichier: `index.php`.

À partir de PrestaShop 1.5, les URL de front office ressemblent à ceci:

- `/index.php?id_category=3&controller=category`
- `/index.php?id_product=7&controller=product`
- `/index.php?id_cms=4&controller=cms`
- etc.

De même, les URL de back office ressemblent à ceci:

- `/admin-dev/index.php?controller=AdminDashboard`
- `/admin-dev/index.php?controller=AdminProducts`
- `/admin-dev/index.php?controller=AdminCmsContent`
- etc.

De plus, le Dispatcher est conçu pour supporter la réécriture d'URL (ou "URL amicales"). Par conséquent, les URL de PrestaShop qui ressemblent à ceci lorsque la réécriture d'URL est désactivée:

- http://myprestashop.com/index.php?controller=category&id_category=3&id_lang=1
- http://myprestashop.com/index.php?controller=product&id_product=1&id_lang=2

... ressemblera à ceci lorsque la réécriture d'URL est activée:

- <http://myprestashop.com/fr/3-music-ipods>
- <http://myprestashop.com/fr/1-ipod-nano.html>

Il y a plusieurs avantages à ce système:

- Il est plus facile d'ajouter un contrôleur.
- Vous pouvez utiliser des itinéraires personnalisés pour modifier vos URL amicales (ce qui est vraiment mieux pour le référencement!)
- Il n'y a qu'un seul point d'entrée dans le logiciel, ce qui améliore la fiabilité de PrestaShop et facilite les développements futurs.

Le répartiteur utilise trois classes abstraites: `Controller`, `FrontController` et `AdminController` (les deux derniers héritant de la première).

De nouvelles routes peuvent être créées en remplaçant la méthode `loadRoutes()`.

L'administrateur du magasin peut modifier l'URL d'un contrôleur à l'aide de la page "SEO & URL" dans le menu "Préférences" du back office.

Contrôleurs dans PrestaShop

Dans une architecture MVC, un contrôleur gère les événements de synchronisation entre la vue et le modèle et les tient à jour. Il reçoit tous les événements de l'utilisateur et déclenche les actions à effectuer.

Si une action a besoin de données à modifier, le Contrôleur "demandera" au Modèle de modifier les données et, à son tour, le Modèle notifiera à la Vue que les données ont été modifiées afin que la Vue puisse se mettre à jour.

Tous les contrôleurs de PrestaShop remplacent en fait la classe Controller par une autre classe héritage:

- `AdminController`,
- `ModuleAdminController`,
- `FrontController`,
- `ModuleFrontController`.

Ils se trouvent dans le dossier / classes / controller.

La classe FrontController

Voici quelques-unes des propriétés de la classe:

Propriété	La description
<code>\$ template</code>	Nom du modèle pour le contenu de la page.
<code>\$ css_files</code>	Liste des fichiers CSS.
<code>\$ js_files</code>	Liste des fichiers JavaScript.
<code>\$ errors</code>	Array des erreurs survenues.
<code>\$ guestAllowed</code>	Le point de savoir si un client qui s'est retiré peut accéder à la page.
<code>\$ initialise</code>	Que la <code>init()</code> fonction ait été appelée.
<code>\$ iso</code>	Le code ISO de la langue actuellement sélectionnée.
<code>\$ n</code>	Le nombre d'articles par page.
<code>\$ orderBy</code>	Le champ utilisé pour trier.
<code>\$ orderBy</code>	Si le tri est ascendant ou descendant ("ASC" ou "DESC").
<code>\$ p</code>	Le numéro de page actuel.
<code>\$ ajax</code>	Si le paramètre ajax est détecté dans la demande, définissez ce drapeau sur true.

Ordre d'exécution des fonctions du contrôleur

1. `__construct()`: Définit toutes les variables des membres du contrôleur.
2. `init()`: Initialise le contrôleur.
3. `setMedia()` ou `setMobileMedia()`: ajoute tous les détails JavaScript et CSS à la page afin qu'ils puissent être combinés, compressés et mis en cache (voir l'outil CCC de PrestaShop, dans la page "Performance" du back office, sous # le menu "Préférences avancées").
4. `postProcess()`: Poignées `ajaxProcess`.
5. `initHeader()`: Appelé avant `initContent()`.
6. `initContent()`: Initialise le contenu.
7. `initFooter()`: Appelé après `initContent()`.
8. `display()` ou `displayAjax()`: affiche le contenu.

Contrôleurs existants du front office

Voici les contrôleurs par défaut et les fichiers de thème qui les utilisent.

Nom du fichier du contrôleur	La description
------------------------------	----------------

AddressController.php	Utilisé par adresse.php pour modifier l'adresse d'un client.
AddressesController.php	Utilisé par adresses.php pour obtenir les adresses des clients.
AttachmentController.php	
AuthController.php	Utilisé par l'authentification .php pour la connexion client.
BestSalesController.php	Utilisé par best-sales.php pour obtenir les best-sellers.
CartController.php	Utilisé par cart.php pour gérer le panier du client.
CategoryController	Utilisé par category.php pour obtenir des catégories de produits.
ChangeCurrencyController.php	
CmsController.php	Utilisé par cms.php pour obtenir une page CMS.
CompareController.php	Utilisé par products-comparison.php pour comparer les produits.
ContactController.php	Utilisé par contact-form.php pour envoyer des messages.
DiscountController.php	Utilisé par discount.php pour obtenir les bons d'achat d'un client.
GetFileController.php	
GuestTrackingController.php	Utilisé par guest-tracking .php pour gérer les commandes des clients.
HistoryController.php	Utilisé par history.php pour obtenir les commandes d'un client.
IdentityController.php	Utilisé par identity.php pour les informations personnelles du client.
IndexController.php	Utilisé par index.php pour afficher la page d'accueil.
ManufacturerController.php	Utilisé par manufacturer.php pour obtenir des fabricants.
MyAccountController.php	Utilisé par my-account.php pour gérer le compte client.
NewProductsController.php	Utilisé par new-products.php pour obtenir de nouveaux produits.
OrderConfirmationController.php	Utilisé par order-confirmation.php pour la confirmation de commande.
OrderController.php	Utilisé par order.php pour gérer le paiement en cinq étapes.
OrderDetailController.php	Utilisé par order-detail.php pour obtenir une commande client.
OrderFollowController.php	Utilisé par order-follow.php pour obtenir les retours d'un client.
OrderOpcController.php	Utilisé par order-opc.php pour gérer la facturation d'une page.
OrderReturnController.php	Utilisé par order-return.php pour obtenir un retour de marchandise.
OrderSlipController.php	Utilisé par order-slip.php pour obtenir les fiches de crédit d'un client.
PageNotFoundController.php	Utilisé par 404.php pour gérer la page "Page non trouvée".
ParentOrderController.php	Gère le code de commande partagé.

PasswordController.php	Utilisé par password.php pour réinitialiser un mot de passe perdu.
PdfInvoiceController.php	
PdfOrderReturnController.php	
PdfOrderSlipController.php	
PrixDropController.php	Utilisé par prices-drop.php pour obtenir des produits à prix réduit.
ProductController.php	Utilisé par product.php pour obtenir un produit.
SearchController.php	Utilisé par search.php pour obtenir les résultats de recherche.
SitemapController.php	Utilisé par sitemap.php pour obtenir le sitemap.
StatisticsController.php	
StoresController.php	Utilisé par stores.php pour obtenir des informations sur les magasins.
SupplierController.php	Utilisé par supplier.php pour obtenir des fournisseurs.

Remplacement d'un contrôleur

Grâce à l'héritage d'objet, vous pouvez modifier les comportements d'un contrôleur ou en ajouter de nouveaux.

Gardez les annulations pour votre propre magasin

Les remplacements dans PrestaShop sont exclusifs. Cela signifie que si votre module remplace l'un des comportements de PrestaShop, un autre module ne pourra pas utiliser correctement ce comportement ou le remplacera de manière prévisible.

Par conséquent, les remplacements ne doivent être utilisés que pour vos propres modules locaux, lorsque vous avez un besoin spécifique qui ne peut être appliqué avec celui-ci.

Il n'est pas recommandé d'utiliser une substitution dans un module que vous souhaitez distribuer (par exemple via le marché PrestaShop Addons), et ils sont interdits dans les modules partenaires.

Comment

Les contrôleurs de PrestaShop sont tous stockés dans le `/controllers` dossier et utilisent le suffixe "Core".

Par exemple, lorsque vous travaillez avec le contrôleur de catégorie:

- Fichier: `/controllers/CategoryController.php`
- Classe: `CategoryControllerCore`

Afin de remplacer un contrôleur, vous devez d'abord créer une nouvelle classe sans le suffixe "Core" et placer son fichier dans le `/override/controllers` dossier.

Par exemple, lors de la suppression du contrôleur de catégorie:

- Fichier: `/override/controllers/front/CategoryController.php`
- Classe: `CategoryController`

Vues dans PrestaShop

PrestaShop utilise le moteur de modèle Smarty pour générer ses vues: <http://www.smarty.net/> .

Vue par thème

Les vues sont stockées dans des `.tpl` fichiers et sont utilisées dans PrestaShop:

- Vue frontale: les fichiers appartiennent au thème activé, qui se trouve dans le `/themes/` dossier.
- Par exemple, avec le thème par défaut: `/themes/default-bootstrap/product.tpl`
- Vue Back Office: les fichiers appartiennent au thème activé, qui se trouve dans le `/admin-dev/themes/` dossier.
- Par exemple, le thème de back office par défaut: `/admin-dev/themes/default/template/controllers/products/information.tpl`

Vue du module

Les modules peuvent ajouter leurs propres modèles pour adapter les parties de l'interface:

- Le front office: `/modules/bankwire/views/templates/front/payment_execution.tpl`
- Le back office: `/modules/blocklayered/views/templates/admin/view.tpl`

Il existe un troisième dossier de modèle, appelé `/hook/`, qui peut être utilisé pour les fichiers de vue liés à un crochet spécifique. Par exemple,

Meilleures pratiques

Un nom de vue est généralement identique au nom du code qui l'utilise. Par exemple, les `404.php` utilisations `404.tpl`.

Remplacer un fichier de vue

Gardez les annulations pour votre propre magasin

Les remplacements dans PrestaShop sont exclusifs. Cela signifie que si votre module remplace l'un des comportements de PrestaShop, un autre module ne pourra pas utiliser correctement ce comportement ou le remplacera de manière prévisible.

Par conséquent, les remplacements ne doivent être utilisés que pour vos propres modules locaux, lorsque vous avez un besoin spécifique qui ne peut être appliqué avec celui-ci.

Il n'est pas recommandé d'utiliser une substitution dans un module que vous souhaitez distribuer (par exemple via le marché PrestaShop Addons), et ils sont interdits dans les modules partenaires.

Comment

Comme il n'y a pas d'héritage, il n'y a aucun moyen de remplacer une vue. Pour modifier une vue, vous devez réécrire le fichier modèle et le placer dans le dossier de votre thème / module, dans le même chemin.

Pour les vues liées à un Helper, vous pouvez utiliser le `/override/` dossier PrestaShop .

Par exemple, si vous souhaitez modifier la façon dont le fichier de modèle de commande de front office: `/admin-dev/themes/default/template/controllers/orders/helpers/view/view.tpl`

... vous devez copier le fichier modèle et son chemin vers le dossier de remplacement:

`/override/controllers/admin/templates/orders/helpers/view/view.tpl`

... puis modifiez le fichier modèle copié pour mieux répondre à vos besoins.

Lorsque vous ajoutez manuellement un fichier de remplacement, n'oubliez pas de supprimer le `/cache/class_index.php` fichier afin que PrestaShop puisse tenir compte de vos modifications.

Consultez la page "Remplacer les comportements par défaut" pour plus d'informations.

Gérer les cookies

PrestaShop utilise des cookies cryptés pour stocker toutes les informations de la session, pour les visiteurs / clients ainsi que pour les employés et les administrateurs.

La classe Cookie (/classes/Cookie.php) est utilisée pour lire et écrire des cookies.

Pour accéder aux cookies à partir du code PrestaShop, vous pouvez l'utiliser:

```
$ this-> context-> cookie;
```

Toutes les informations stockées dans un cookie sont disponibles en utilisant ce code:

```
$ this-> context-> cookie-> variable;
```

Si vous devez accéder au cookie PrestaShop à partir du code non-PrestaShop, vous pouvez utiliser:

```
include_once ('path_to_prestashop / config / config.inc.php');
include_once ('path_to_prestashop / config / settings.inc.php');
include_once ('path_to_prestashop / classes / Cookie.php');
$ cookie = nouveau Cookie ('ps'); // Utilisez "psAdmin" pour lire le cookie d'un employé.
```

Données stockées dans un cookie de client / client

Jeton	La description
date_a_jj	La date et l'heure où le cookie a été créé (en AAAA-MM-DD HH: MM: format SS).
id_lang	L'ID de la langue sélectionnée.
id_currency	L'ID de la devise sélectionnée.
last_visited_category	il identifie la dernière catégorie de listes de produits visitées.
ajax_blockcart_display	Que le bloc de panier soit "étendu" ou "effondré".
vu	Les ID des produits récemment regardés comme une liste séparée par virgule.
id Wishlist	L'identifiant de la liste de diffusion actuelle s'affiche dans le bloc de sélection.
TOS contrôlé	La case à cocher "Conditions d'utilisation" a-t-elle été cochée (1 si elle a eu et 0 si elle n'a pas).
id_guest	L'ID invité du visiteur lorsqu'il n'est pas connecté.
id_connection	L'ID de connexion de la session en cours du visiteur.
id_customer	L'ID du client du visiteur lors de sa connexion.
customer_last_name	Le nom de famille du client.
customer_firstname	Le prénom du client.
connecté	Que le client soit connecté.
passwd	Le hasard MD5 du mot de passe utilisé par le client pour se connecter, COOKIE_KEY `` in `` config/settings.inc.php
email	L'adresse courriel que le client avait l'habitude de se connecter.
id_cart	L'ID du panier actuel affiché dans le bloc de panier.
checksum	La somme de contrôle de Blowfish utilisée pour déterminer si le cookie a été modifié par un tiers. Le client sera déconnecté et le cookie supprimé si la somme de contrôle ne correspond pas.

Données stockées dans un cookie de l'employé / administrateur

Jeton	La description

date_a jj	La date et l'heure où le cookie a été créé (en AAAA-MM-DD HH: MM: format SS).
id_lan g	L'ID de la langue sélectionnée.
id_emp loyee	L'identité de l'employé.
nom de famille	Le nom de famille de l'employé.
Prénom	Le prénom de l'employé.
email	L'adresse électronique utilisée par l'employé pour se connecter.
profil	L'ID du profil qui détermine les onglets auxquels l'employé peut accéder.
passwd	Le hasard MD5 du mot de passe utilisé par l'employé pour se connecter. <code>_COOKIE_KEY_ `` in ``config/setti ngs.inc.php</code>
checksu m	La somme de contrôle de Blowfish utilisée pour déterminer si le cookie a été modifié par un tiers. Si la somme de contrôle ne correspond pas, le client sera déconnecté et le cookie est supprimé.

Gérer les crochets

Les crochets sont un moyen d'associer votre code à certains événements spécifiques de PrestaShop.

La plupart du temps, ils sont utilisés pour insérer du contenu dans une page.

Par exemple, la page d'accueil du thème par défaut de PrestaShop comporte les crochets suivants:

Nom de crochet	La description
displayHeader	Affiche le contenu dans la zone d'en-tête de la page.
displayTop	Affiche le contenu dans la zone supérieure de la page.
AfficherLeftColumn	Affiche le contenu dans la colonne de gauche de la page.
afficherHome	Affiche le contenu dans la zone centrale de la page.
displayRightColumn	Affiche le contenu dans la colonne de droite de la page.
displayFooter	Affiche le contenu dans la zone de bas de page de la page.

Les crochets peuvent également être utilisés pour effectuer des actions spécifiques dans certaines circonstances (c.-à-d. Envoi d'un courrier électronique au client).

Vous pouvez obtenir une liste complète des crochets disponibles dans PrestaShop 1.6 dans le chapitre "Hooks in PrestaShop 1.6" du Guide du développeur.

Utilisation des crochets

... dans un contrôleur

Il est facile d'appeler un crochet à partir d'un contrôleur: il vous suffit d'utiliser son nom avec la `hookExec()` méthode: `Module::hookExec('NameOfHook');`

Par exemple:

```
$ this-> context-> smarty-> assign ('HOOK_LEFT_COLUMN', Module :: hookExec ('displayLeftColor
```

... dans un module

Pour attacher votre code à un crochet, vous devez créer une méthode publique non statique, en commençant par le mot-clé "crochet" suivi de "affichage" ou "action" et du nom du crochet que vous souhaitez utiliser.

Cette méthode reçoit un (et un seul) argument: un tableau des informations contextuelles envoyées au crochet.

```
Public function hookDisplayNameOfHook ($ params)
{
    // Votre code.
}
```

Pour qu'un module réponde à un appel de crochet, le crochet doit être enregistré dans PrestaShop. L'enregistrement des crochets est effectué en utilisant la `registerHook()` méthode. L'inscription se fait généralement lors de l'installation du module.

```
installation de la fonction publique ()
{
    return parent :: install () && $ this-> registerHook ('NameOfHook');
}
```

... dans un thème

Il est facile d'appeler un hook à partir d'un fichier modèle (`.tpl`): vous devez simplement utiliser son nom avec la fonction hook. Vous pouvez ajouter le nom d'un module que vous voulez que le hook soit exécuté.

Par exemple:

```
{ hook h = 'displayLeftColumn' mod = 'blockcart' }
```

Création de votre propre crochet

Vous pouvez créer de nouveaux crochets PrestaShop en ajoutant un nouvel enregistrement dans le `ps_hook` tableau dans votre base de données MySQL. Vous pourriez le faire de manière difficile:

```
INSERT INTO `ps_hook` (` name`, `title`, ` description`) VALUES ('nameOfHook', 'The name of y
```

... mais PrestaShop vous permet de le faire de manière simple:

```
$ this-> registerHook ('NameOfHook');
```

Si le crochet "NameOfHook" n'existe pas, PrestaShop le créera pour vous. Il n'est plus nécessaire de faire la requête SQL.

Qu'est-ce qu'un module PrestaShop?

L'extensibilité de PrestaShop tourne autour des modules, qui sont de petits programmes qui utilisent les fonctionnalités de PrestaShop et les modifient ou les ajoutent afin de rendre PrestaShop plus facile à utiliser ou plus adapté aux besoins du marchand.

Principes techniques derrière un module

Un module PrestaShop se compose d'un fichier PHP principal avec autant d'autres fichiers PHP que nécessaire, ainsi que les fichiers et les éléments matériels (images, JavaScript, CSS, etc.) nécessaires pour afficher l'interface du module, que ce soit pour le client (sur le front office) ou au marchand (sur le back office).

Tout module PrestaShop, une fois installé sur un magasin en ligne, peut interagir avec un ou plusieurs "crochets". Les crochets vous permettent d'accrocher / attacher votre code à la vue actuelle au moment de l'analyse du code (c.-à-d. Lors de l'affichage du panier ou de la fiche produit, lors de l'affichage du stock actuel, etc.). Plus précisément, un crochet est un raccourci vers les différentes méthodes disponibles à partir de l'objet, tel qu'il est attribué à ce crochet.

Principes de fonctionnement des modules

Les modules sont le moyen idéal pour que votre talent et votre imagination, en tant que développeur, s'expriment, car les possibilités créatives sont nombreuses et vous pouvez faire n'importe quoi avec l'API du module PrestaShop.

Tout module:

- peut afficher une variété de contenu (blocs, texte, etc.), effectuer de nombreuses tâches (mise à jour par lot, importation, exportation, etc.), interface avec d'autres outils, et bien plus encore.
- peut être configuré selon les besoins; plus il est configurable, plus il sera facile à utiliser, et sera donc capable de répondre aux besoins d'une plus large gamme d'utilisateurs.
- peut ajouter des fonctionnalités à PrestaShop sans avoir à modifier ses fichiers de base, ce qui facilite l'exécution d'une mise à jour de PrestaShop sans avoir transposé tous les changements de base. En effet, vous devez toujours vous efforcer de vous éloigner des fichiers de base lors de la construction d'un module, même si cela peut sembler nécessaire dans certaines situations.

Principales différences entre 1,6 et 1,7 modules

PrestaShop 1.7 a été construit de sorte que les modules qui ont été écrits pour PS 1.6 puissent fonctionner presque tel quel, sauf pour des modifications mineures et une mise à jour de cosmétiques, les fichiers de modèle ayant besoin d'être adaptés au thème 1.7 par défaut.

Les modifications majeures apportées au développement de modules dans PrestaShop 1.7 sont expliquées en détail [dans cet article de construction](#) et sont intégrées dans cette documentation mise à jour. Si vous savez déjà comment créer un module qui fonctionne avec PS 1.6, nous vous conseillons fortement de lire cet article de haut en bas afin de vous lancer dans le développement 1.7.

Structure de fichier pour un module PS 1.7

FAIRE

La structure du fichier PrestaShop 1.7

Les développeurs de PrestaShop ont fait de leur mieux pour séparer de manière claire et intuitive les différentes parties du logiciel.

Voici comment les fichiers sont organisés:

- **/admin**(le nom est personnalisé lors de l'installation): contient tous les fichiers PrestaShop relatifs au back office. Lorsque vous accédez à ce dossier avec votre navigateur, vous devrez fournir une identification appropriée, pour des raisons de sécurité. **Important** : vous devez vous assurer de protéger ce dossier avec un fichier **.htaccess** ou **.htpasswd**!
- **/app**: NOUVEAU EN 1.7.

- **/cache**: contient des dossiers temporaires qui sont générés et réutilisés afin d'alléger la charge du serveur.
- **/classes**: contient tous les fichiers appartenant au modèle d'objet de PrestaShop (certains sont utilisés pour le front office, d'autres pour le back office). Chaque fichier représente (et contient) une classe PHP, et ses méthodes / propriétés.
- **/config**: contient tous les fichiers de configuration de PrestaShop. Sauf demande, vous *ne devriez jamais* les éditer, car ils sont directement gérés par l'installateur et le back office de PrestaShop.
- **/controllers**: contient tous les fichiers appartenant aux contrôleurs PrestaShop - comme dans Model-View-Controller (ou MVC), l'architecture logicielle utilisée par PrestaShop. Chaque fichier contrôle une partie spécifique de PrestaShop.
- **/docs**: contient de la documentation, des licences et des exemples de fichiers d'importation. *Remarque* : il devrait être supprimé dans un environnement de production.
- **/download**: contient vos produits virtuels, qui peuvent être téléchargés par l'utilisateur qui l'a payé. Les fichiers sont stockés avec un nom de fichier md5.
- **/img**: contient toutes les images, icônes et fichiers image par défaut de PrestaShop, c'est-à-dire ceux qui n'appartiennent pas au thème. C'est là que vous pouvez trouver les images pour les catégories de produits (**/c** sous-dossier), celles pour les produits (**/psous**-dossier), et celles pour le back office lui-même (**/admin**sous-dossier).
- **/install**: contient tous les fichiers liés au programme d'installation de PrestaShop. Vous devrez le supprimer après l'installation, afin d'augmenter la sécurité.
- **/js**: contient tous les fichiers JavaScript qui ne sont pas liés aux thèmes. La plupart d'entre eux appartiennent au back office. C'est aussi là que vous trouverez le framework jQuery.
- **/localization**: contient tous les fichiers de localisation de PrestaShop, c'est-à-dire les fichiers qui contiennent des informations locales telles que la devise, la langue, les règles fiscales et les groupes de règles fiscales, les états et les différentes unités utilisées dans le pays choisi (volume, litre, poids kilogrammes, etc.).
- **/mails**: contient tous les fichiers HTML et texte relatifs aux e-mails envoyés par PrestaShop. Chaque langue a son dossier spécifique, où vous pouvez modifier manuellement leur contenu si vous le souhaitez. PrestaShop contient un outil pour éditer vos e-mails, situés dans le back office, dans la page Localisation> Traduction.
- **/modules**: contient tous les modules de PrestaShop, chacun dans son propre dossier. Si vous souhaitez retirer définitivement un module, commencez par le désinstaller à partir du back-office, puis seulement si vous supprimez son dossier.
- **/override**: il s'agit d'un dossier spécial qui est apparu avec PrestaShop 1.4. En utilisant la convention régulière de dossier / filename de PrestaShop, il est possible de créer des fichiers qui remplacent les classes ou les contrôleurs par défaut de PrestaShop. Cela vous permet de modifier le comportement de base de PrestaShop sans toucher aux fichiers d'origine, en les gardant en sécurité pour la prochaine mise à jour. *Remarque* : les remplacements ne sont pas recommandés pour les modules que vous souhaitez distribuer / vendre, et sont strictement interdits dans les modules partenaires. Gardez-les pour votre propre magasin.
- **/pdf**: contient tous les fichiers de modèle (**.tpl**) relatifs à la génération de fichier PDF (facture, feuillets de livraison, etc.). Changez ces fichiers afin de modifier l'apparence des fichiers PDF que PrestaShop génère.
- **/src**: NOUVEAU EN 1.7. Contient les fichiers d'architecture, comprenant le cadre Symfony, le cadre hérité et les classes d'adaptateur.
- **/tests**: NOUVEAU EN 1.7. Contient des tests automatisés. Ce dossier ne fait pas partie des archives publiques.
- **/themes**: contient tous les thèmes actuellement installés, chacun dans son propre dossier.
- **/tools**: contient des outils externes intégrés dans PrestaShop. Par exemple, vous trouverez Smarty (moteur de modèle), TCPDF (générateur de fichier PDF), Swift (expéditeur de messagerie), PEAR XML Parser (outil PHP), etc.
- **/translations**: contient un sous-dossier pour chaque langue disponible. Toutefois, si vous souhaitez modifier la traduction, vous devez l'utiliser à l'aide de l'outil interne PrestaShop, et ne pas les modifier directement dans ce dossier.
- **/travis-scripts**: NOUVEAU EN 1.7. Contient des scripts Travis-spezifc. Ce dossier ne fait pas partie des archives publiques.
- **/upload**: contient les fichiers qui seront téléchargés par les clients pour des produits personnalisables (par exemple, une image qu'un client souhaite imprimer sur une tasse).
- **vendor**: NOUVEAU EN 1.7. Contient divers outils et cadres tiers utilisés par PrestaShop, tels que Composer, CURL, Doctrine, etc.
- **web**: NOUVEAU EN 1.7. Contient différents fichiers spécifiques au serveur Web, tels que favicon.ico ou robots.txt. Ce dossier ne fait pas partie des archives publiques.
- **/webservice**: contient des fichiers qui permettent aux applications tierces d'accéder à PrestaShop via son API.

Dossiers racine qui ont été supprimés entre 1,6 et 1,7:

- **/css**: contenait tous les fichiers CSS qui ne sont pas attachés aux thèmes; par conséquent, ils étaient principalement utilisés par le back-office de PrestaShop et ont maintenant été transférés dans le **theme.css**dossier dans le **/admin/themes/new-theme/public**dossier.

- `/log`: contient les fichiers journaux générés par PrestaShop à différentes étapes, par exemple pendant la procédure d'installation.

Dossier Modules

Les modules de PrestaShop se trouvent dans le `/modules` dossier, qui est à la racine du dossier principal de PrestaShop. Ceci est vrai pour les modules par défaut (fournis avec PrestaShop) et les modules tiers qui sont ensuite installés.

Les modules peuvent également faire partie d'un thème s'ils sont vraiment spécifiques. Dans ce cas, ils seraient dans le `/modules` dossier propre du thème, et donc sous le chemin suivant: `/themes/[my-theme]/modules`

Chaque module possède son propre sous-dossier dans le `/modules` dossier: `/bankwire` `/birthdaypresent`, etc. À propos du cache

Le `/cache/class_index.php` fichier contient le lien entre la classe et le fichier de déclaration. S'il y a un problème de mise en cache, ce fichier peut être supprimé en toute sécurité.

Le `/config/xml` dossier contient la liste de tous les modules de base:

```
default_country_modules_list . xml
must_have_modules_list . xml
tab_modules_list . xml
```

Lorsque le front-end du magasin ne reflète pas vraiment vos modifications et que la vidange du cache du navigateur n'est pas efficace, essayez de vider les dossiers suivants:

```
/ cache / smarty / cache
/ cache / smarty / compile
```

Création d'un premier module

Structure de fichier pour un module PrestaShop

Un module est constitué de beaucoup de fichiers, tous stockés dans un dossier portant le même nom que le module, ce dossier étant à son tour stocké dans le dossier / modules à la racine du dossier principal de PrestaShop: / modules / name_of_the_module /.

Voici les fichiers et dossiers possibles pour un module PrestaShop 1.6:

Nom du fichier / dossier	La description	Détails
name_of_the_module.php	Fichier principal.	Le fichier PHP principal devrait avoir le même nom que le dossier racine du module. Par exemple, pour le module BlockCMS: Nom du dossier: / modules / blockcms Nom du fichier principal: / modules / blockcms / blockcms.php
config.xml	Fichier de configuration du cache.	S'il n'existe pas encore, ce fichier est automatisé généré par PrestaShop lorsque le module est installé pour la première fois.
logo.gif ou logo.jpg (jusqu'à v1.4) logo.png (v1.5 +)	Fichiers d'icônes représentant ce module dans le back office.	PrestaShop 1.4: 16 16 pixels Gif ou Jpeg. PrestaShop 1.5: 32 Fichier PNG de 32 pixels. Si votre module fonctionne à la fois sur PrestaShop 1.4 et PrestaShop 1.5+, vous devriez avoir à la fois un logo.gif ET un fichier logo.png.
/ vues	Ce dossier contient les fichiers View.	
/ vues / modèles	Ce dossier contient les fichiers de modèle de votre module (.tpl).	Si le module doit fonctionner avec PrestaShop 1.4, les fichiers modèles doivent être placés soit directement à la racine du dossier du module, soit dans un dossier / template à la racine du dossier du module.
/ views / templates / admin	Sous-dossier pour les fichiers modèles utilisés par les contrôleurs d'administration du module.	
/ views / templates / front	Sous-dossier pour les fichiers modèles utilisés par les contrôleurs du module frontal du module.	
/ views / templates / hook	Sous-dossier pour les fichiers modèles utilisés par les crochets du module.	

Nom du fichier / dossier	La description	Détails
/ views / css	Sous-dossier pour les fichiers CSS utilisés.	Si le module doit fonctionner avec PrestaShop 1.4, les fichiers CSS doivent être placés dans un dossier / css à la racine du dossier du module.
/ views / js	Sous-dossier pour les fichiers JavaScript.	Si le module doit fonctionner avec PrestaShop 1.4, les fichiers JavaScript doivent être placés dans un dossier / js à la racine du dossier du module.
/ views / img	Sous-dossier pour les fichiers d'image.	Si le module doit fonctionner avec PrestaShop 1.4, les fichiers image doivent être placés dans un dossier / img à la racine du dossier du module.
/ contrôleurs	Ce dossier contient les fichiers Controller. Vous pouvez utiliser les mêmes chemins de sous-dossiers que pour les fichiers View. Par exemple, / modules / bankwired / controllers / front / payment.php.	
/passer outre	Sous-dossier pour le code classe-override.	Ceci est très utile lorsque vous devez modifier certains des codes PrestaShop par défaut. Comme vous ne devez pas le faire, vous pouvez remplacer le code par défaut. Par exemple, / modules / google / override / classes / Shop.php étend la classe ShopCore par défaut.
/ traductions	Sous-dossier pour les fichiers de traduction.	fr.php, en.php, es.php, etc.
/ themes / [theme_name] / modules	Sous-dossier pour remplacer les fichiers .tpl et les fichiers de langues, si nécessaire.	Ce dossier est essentiel lors des modifications d'un module existant, afin de pouvoir l'adapter sans avoir à toucher ses fichiers d'origine. Notamment, il vous permet de gérer les fichiers modèles du module de diverses façons, selon le thème actuel.
/surclassement	Sous-dossier pour les fichiers de mise à niveau	Lors de la publication d'une nouvelle version du module, l'ancien peut avoir besoin d'une mise à jour de ses données ou de ses fichiers. Cela peut être fait en utilisant ce dossier.

Seuls les trois premiers sont nécessaires pour un module de base: le fichier principal, le fichier de configuration de cache (qui est autogénéré par PrestaShop de toute façon) et le fichier d'icônes. Tous les autres peuvent être utilisés si nécessaire, mais un module peut fonctionner sans eux.

Le module peut aussi que beaucoup d'autres fichiers et dossiers au besoin: `/css`, `/img`, `/js`, etc.

Bibliothèques externes

Si vous souhaitez utiliser une bibliothèque externe, elle doit être placée dans un dossier dédié.

Ce dossier peut utiliser l'un de ces noms: «lib», «libs», «bibliothèques», «sdk», «fournisseur», «fournisseurs».

Choisissez le plus approprié pour votre bibliothèque (en effet, les «bibliothèques» n'ont pas la même signification que «sdk»). Vous pouvez avoir plus d'un tel dossier, par exemple `/libet /sdk`.

La convention de codage PrestaShop

Avant de commencer à écrire le code pour votre module PrestaShop, vous devez savoir que l'équipe PrestaShop utilise un ensemble spécifique de conventions de codage (ou normes de codage, norme de codage, etc.).

Comme le dit Wikipedia, «Les conventions de codage sont un ensemble de lignes directrices pour un langage de programmation spécifique qui recommandent le style de programmation, les pratiques et les méthodes pour chaque aspect d'un programme écrit dans cette langue» et «Bonnes procédures, bonne méthodologie et bonnes normes de codage peut être utilisé pour générer un projet de sorte que la qualité soit optimisée et que le temps de développement global et les coûts de développement et de maintenance soient réduits au minimum.» (Voir http://fr.wikipedia.org/wiki/Coding_conventions).

Les normes de PrestaShop sont disponibles sur cette page: <http://doc.prestashop.com/display/PS16/Coding+Standards> . Vous devriez le lire afin de vous assurer que le code que vous produisez correspond correctement au code global du projet PrestaShop. Le PHPCodeSniffer peut vous aider à vous assurer de suivre correctement la norme.

Création d'un premier module

Créons un premier module simple; cela nous permettra de mieux décrire sa structure. Nous le nommerons «Mon module».

Tout d'abord, créez le dossier du module, dans le `/modules` dossier. Il devrait avoir le même nom que le module, sans espace, seuls les caractères alphanumériques, le trait d'union et le trait de soulignement, tout en minuscules: `/mymodule`.

Ce dossier doit contenir le fichier principal, un fichier PHP du même nom que le dossier, qui se chargera de la majeure partie du traitement: `mymodule.php`.

Cela suffit pour un module très basique, mais évidemment, d'autres fichiers et dossiers peuvent être ajoutés plus tard.

Le test constant

Le fichier `mymodule.php` principal doit commencer par le test suivant:

```
<? php
si (! défini ('_ PS_VERSION_'))
{
    Sortie;
}
```

Cela vérifie l'existence d'une constante PrestaShop toujours existante (son numéro de version), et si elle n'existe pas, elle empêche le chargement du module. Le seul but est d'empêcher les visiteurs malveillants de charger ce fichier directement.

Notez que, comme l'exigent les normes de codage PrestaShop (voir ci-dessus), nous n'utilisons pas de balise de fermeture PHP.

La classe principale

Le fichier principal doit contenir la classe principale du module (avec d'autres classes si nécessaire). PrestaShop utilise une programmation orientée objet, tout comme ses modules.

Cette classe doit porter le même nom que le module et son dossier, dans CamelCase (voir <http://en.wikipedia.org/wiki/Camel-Case>). Dans notre exemple: `MyModule`. En outre, cette classe doit étendre la `Module` classe, afin d'hériter toutes ses méthodes et

ses attributs.

```
<? php
si (! défini ('_ PS_VERSION_'))
{
    Sortie;
}

classe MyModule extends Module
{
}
```

Il peut tout aussi bien étendre une classe dérivée de module, pour des besoins spécifiques: **PaymentModule**, **ModuleGridEngine**, **ModuleGraph**, etc.

À ce stade, si vous placez le dossier du module dans le dossier / modules, le module peut déjà être vu dans la page "Modules" dans le back office, dans la section "Autres modules" - mais sans nom réel ni miniature.

La méthode constructeur

Maintenant, remplissons le bloc de code de la classe avec les lignes de constructeur essentielles. Un constructeur est une fonction dans une classe qui est automatiquement appelée lorsque vous créez une nouvelle instance d'une classe avec une nouvelle. Dans le cas d'un PrestaShop, la classe constructeur est la première méthode à appeler lorsque le module est chargé par PrestaShop. C'est donc le meilleur endroit pour définir la plupart de ses détails.

```
<? php
si (! défini ('_ PS_VERSION_'))
{
    Sortie;
}

classe MyModule extends Module
{
    fonction publique __construct ()
    {
        $ this-> name = 'mymodule';
        $ this-> tab = 'front_office_features';
        $ this-> version = '1.0.0';
        $ this-> author = 'Firstname Lastname';
        $ this-> need_instance = 0;
        $ this-> ps_versions_compliancy = array ('min' => '1.6', 'max' => _PS_VERSION_);
        $ this-> bootstrap = true;

        parent :: __ construire ();

        $ this-> displayName = $ this-> l ('Mon module');
        $ this-> description = $ this-> l ('Description de mon module. ');

        $ this-> confirmUninstall = $ this-> l ('Êtes-vous sûr de vouloir désinstaller?');

        si (! Configuration :: get ('MYMODULE_NAME'))
            $ this-> warning = $ this-> l ('Aucun nom fourni');
    }
}
```

Examinons chaque ligne de cette première version de la **MyModule** classe ...

```
fonction publique __construct ()

Cette ligne définit la fonction constructeur de la classe.
$ this-> name = 'mymodule';
$ this-> tab = 'front_office_features';
$ this-> version = '1.0';
$ this-> author = 'Firstname Lastname';
```

Cette section attribue une poignée d'attributs à l'instance de classe (**this**):

- Attribut «nom». Ces attributs servent d'identifiant interne. La valeur DOIT être le nom du dossier du module. N'utilisez pas de caractères ou d'espaces spéciaux, et gardez-le en minuscules.
- Attribut 'onglet'. Le titre de la section qui doit contenir ce module dans la liste des modules Back Office de PrestaShop. Vous pouvez utiliser un nom existant, tel que `seo`, `front_office_features` ou `analytics_stats`, ou personnalisé. Dans ce dernier cas, une nouvelle section sera créée avec votre identifiant. Nous avons choisi "`front_office_features`" car ce premier module aura principalement un impact sur le front-end.
- Attribut 'version'. Le numéro de version du module, affiché dans la liste des modules. C'est une chaîne, afin que vous puissiez utiliser une variation telle que «1.0b», «3.07 beta 3» ou «0.94 (pas pour l'utilisation de la production)».
- Attribut «auteur». Ceci est affiché tel quel dans la liste des modules PrestaShop.

Voici la liste des attributs "Tab" disponibles et leur section correspondante dans la page "Modules":

Attribut "Tab"	Section du module
administration	Administration
marketing publicitaire	Marketing publicitaire
analytics_stats	Analyse et statistiques
facture de facturation	Facturation et factures
check-out	Check-out
gestion de contenu	Gestion de contenu
tableau de bord	Tableau de bord
envoi par courrier électronique	E-mail
exportation	Exportation
front_office_features	Fonctionnalités Front Office
i18n_localization	I18n & Localisation
market_place	Place du marché
merchandising	Merchandising
migration_tools	Outils de migration
mobile	Mobile
autres	Autres modules
payment_gateways	Paiements et passerelles
sécurité de paiement	Sécurité des paiements
pricing_promotion	Prix et promotion
quick_bulk_update	Mise à jour rapide / en vrac
search_filter	Recherche et filtre
Seo	SEO
shipping_logistics	Expédition et logistique
diaporamas	Diaporamas
smart_shopping	Smart Shopping
réseaux sociaux	Réseaux sociaux

Continuons avec la ligne suivante dans ce bloc de code:

```
$ this-> need_instance = 0;
$ this-> ps_versions_compliancy = array ('min' => '1.5', 'max' => '1.6');
$ this-> bootstrap = true;
```

Cette section gère la relation avec le module et son environnement (à savoir, PrestaShop):

- `need_instance`. Indique s'il faut charger la classe du module lors de l'affichage de la page "Modules" dans le back office. Si cette option est définie à 0, le module ne sera pas chargé et, par conséquent, dépensera moins de ressources pour générer la page "Modules". Si votre module doit afficher un message d'avertissement dans la page "Modules", vous devez définir cet attribut sur 1.

- *ps_versions_comliancy* . Indique la version de ce module compatible avec PrestaShop. Dans l'exemple ci-dessus, nous écrivons explicitement que ce module ne fonctionnera que avec PrestaShop 1.5.x et aucune autre version majeure.
- *bootstrap* . Indique que les fichiers de modèle du module ont été construits avec les outils de démarrage de PrestaShop 1.6 à l'esprit - et, par conséquent, que PrestaShop ne devrait pas essayer d'envelopper le code de modèle pour l'écran de configuration (s'il en existe un) avec des balises helper.

Ensuite, nous appelons la méthode constructeur à partir de la classe PHP parent:

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

Cela déclenchera de nombreuses actions de PrestaShop auxquelles vous n'avez pas besoin de connaître à ce stade. L'appel de la méthode du constructeur parental doit être effectué après la création et avant toute utilisation de la méthode de traduction.`this->name variablethis->l()`

La section suivante traite des chaînes de texte, qui sont encapsulées dans la méthode de traduction de PrestaShop `l()`:

```
$ this-> displayName = $ this-> l ('Mon module');
$ this-> description = $ this-> l ('Description de mon module. ');

$ this-> confirmUninstall = $ this-> l ('Êtes-vous sûr de vouloir désinstaller?');

si (! Configuration :: get ('MYMODULE_NAME'))
    $ this-> warning = $ this-> l ('Aucun nom fourni.');
```

Ces lignes attribuent respectivement:

- Un nom pour le module, qui sera affiché dans la liste des modules du back office.
- Une description pour le module, qui sera affichée dans la liste des modules du back office.
- Un message, en demandant à l'administrateur s'il désire vraiment désinstaller le module. Pour être utilisé dans le code d'installation.
- Un avertissement que le module n'a pas encore sa `MYMODULE_NAME` valeur de base de données (ce dernier point étant spécifique à notre exemple, comme nous le verrons plus loin).

La méthode constructeur est maintenant terminée. Vous pouvez ajouter plus d'informations plus tard si nécessaire, mais c'est le strict minimum pour un module de travail.

Accédez maintenant à votre back-office, dans la page Modules: le module est visible dans la liste des modules, avec ses informations affichées - et aucune icône pour l'instant.

Vous pouvez installer le module, mais il ne fait rien encore.

Lorsque vous cliquez sur le bouton "Installer" pour votre module, il affichera une fenêtre de module indiquant que votre module n'est pas approuvé.

La seule façon de rendre votre module Trusted est de le distribuer via le marché PrestaShop Addons (avec une clé d'identification unique) ou de devenir un partenaire PrestaShop. D'autres modules approuvés sont les indigènes.

Pour installer le module, cliquez sur "Continuer avec l'installation" sur cet écran.

méthodes d'installation () et de désinstallation ()

Certains modules ont plus de besoins que d'utiliser simplement les fonctionnalités de PrestaShop de manière spéciale. Votre module pourrait avoir besoin d'effectuer des actions lors de l'installation, telles que vérifier les paramètres de PrestaShop ou enregistrer ses propres paramètres dans la base de données. De même, si vous avez changé de choses dans la base de données lors de l'installation, il est fortement recommandé de les modifier (ou de les supprimer) lors de la désinstallation du module.

Les `install()` et `uninstall()` méthodes permettent de contrôler ce qui se passe lorsque l'administrateur du magasin installe ou désinstalle le module. Ils doivent être inclus dans le bloc de code de la classe principale (dans notre exemple, la `MyModule` classe) - au même niveau que la méthode du constructeur.

La méthode install ()

Voici le strict minimum pour la `install()` méthode:

```
installation de la fonction publique ()
{
    si (! parent :: install ())
        renvoyer faux;
    retourner vrai;
}
```

Dans cette première et extrêmement simpliste incarnation, cette méthode nécessite le minimum requis: `return true` renvoyé par la `install()` méthode du module, qui retourne soit `true` si le module est correctement installé, soit `false` autrement. En l'état, si nous n'avions pas créé cette méthode, la méthode de la superclasse aurait été appelée au lieu de cela, rendant le résultat final identique. Néanmoins, nous devons mentionner cette méthode, car il sera très utile une fois que nous devons effectuer des vérifications et des actions pendant le processus d'installation du module: créer des tables SQL, copier des fichiers, créer des variables de configuration, etc.

Ainsi, par exemple, comment vous pouvez étendre la `install()` méthode pour effectuer des vérifications d'installation. Dans l'exemple suivant, nous effectuons les tâches suivantes lors de l'installation:

- Vérifiez que la fonctionnalité Multistore est activée et, dans l'affirmative, définissez le contexte actuel dans tous les magasins de cette installation de PrestaShop.
- Vérifiez que la classe parent du module est installée.
- Vérifiez que le module peut être attaché au `leftColumn` crochet.
- Vérifiez que le module peut être attaché au `header` crochet.
- Créez le `MYMODULE_NAME` paramètre de configuration, définissant sa valeur sur «mon ami».

```
installation de la fonction publique ()
{
    si (Shop :: isFeatureActive ())
        Shop :: setContext (Shop :: CONTEXT_ALL);

    si (! parent :: install () ||
        ! $ this-> registerHook ('leftColumn') ||
        ! $ this-> registerHook ('header') ||
        ! Configuration :: updateValue ('MYMODULE_NAME', 'mon ami')
    )
        renvoyer faux;

    retourner vrai;
}
```

Si l'une des lignes du bloc de test échoue, la méthode revient `false` et l'installation ne se produit pas.

La méthode de désinstallation ()

Voici le strict minimum pour la méthode de désinstallation ():

```
fonction publique désinstaller ()
{
    si (! parent :: uninstall ())
        renvoyer faux;
    retourner vrai;
}
```

S'appuyant sur cette base, nous voulons une méthode de désinstallation () qui supprime les données ajoutées à la base de données lors de l'installation (`MYMODULE_NAME` configuration). Cette méthode ressemblerait à ceci:

```
fonction publique désinstaller ()
{
    si (! parent :: uninstall () ||
        ! Configuration :: deleteByName ('MYMODULE_NAME')
    )
        renvoyer faux;
}
```

```
    retourner vrai;
}
```

L'objet Configuration

Comme vous pouvez le voir, nos trois blocs de code (`__construct()`, `install()` et `uninstall()`) utilisent tout d'un nouvel objet, `Configuration`.

Il s'agit d'un objet spécifique à PrestaShop, conçu pour aider les développeurs à gérer leurs paramètres de module. Il stocke ces paramètres dans la base de données de PrestaShop sans nécessiter d'utilisation de requêtes SQL. Plus précisément, cet objet gère les données de la `ps_configuration` table de la base de données.

Les principales méthodes

Jusqu'à présent, nous avons utilisé trois méthodes, auxquelles nous allons ajouter un quatrième dans la liste ci-dessous:

- `Configuration::get('myVariable')`: récupère une valeur spécifique à partir de la base de données.
- `Configuration::getMultiple(array('myFirstVariable', 'mySecondVariable', 'myThirdVariable'))`: récupère plusieurs valeurs de la base de données et renvoie un tableau PHP.
- `Configuration::updateValue('myVariable', $value)`: met à jour une variable de base de données existante avec une nouvelle valeur. Si la variable n'existe pas encore, elle la crée avec cette valeur.
- `Configuration::deleteByName('myVariable')`: supprime la variable de base de données.

Il y en a beaucoup plus, comme `getInt()` ou `hasContext()`, mais ces quatre sont ceux que vous utiliserez le plus.

Notez que lors de l'utilisation `updateValue()`, le contenu de la valeur `$` peut être n'importe quoi, qu'il s'agisse d'une chaîne, d'un nombre, d'un système PHP sérialisé ou d'un objet JSON. Tant que vous avez correctement codé la fonction de gestion des données, tout va bien. Par exemple, voici comment gérer un tableau PHP en utilisant l' `Configuration` objet:

```
// Enregistrement d'un tableau sérialisé.
Configuration::updateValue('MYMODULE_SETTINGS', sérialisation(array(true, true, false)));

// Récupération du tableau.
$configuration_array = unserialize(Configuration::get('MYMODULE_SETTINGS'));
```

Comme vous pouvez le voir, ceci dans un objet très utile et facile à utiliser, et vous l'utiliserez certainement dans de nombreuses situations. La plupart des modules natifs l'utilisent aussi pour leurs propres paramètres.

Gestion de la fonctionnalité Multistore

Par défaut, toutes ces méthodes fonctionnent dans les limites du contexte actuel du magasin, que PrestaShop utilise ou non la fonction multistore.

Cependant, il est possible de travailler en dehors du contexte actuel et d'avoir impact sur d'autres magasins connus. Cela se fait à l'aide de trois paramètres facultatifs, qui ne sont pas présentés dans la liste ci-dessus:

- `id_lang`: vous permet de forcer la langue avec laquelle vous voulez travailler.
- `id_shop_group`: vous permet d'indiquer le groupe de magasins du magasin cible.
- `id_shop`: vous permet d'indiquer l'identifiant du magasin cible.

Par défaut, ces trois paramètres utilisent les valeurs du contexte actuel, mais vous pouvez les utiliser pour cibler les autres magasins.

Notez qu'il n'est pas recommandé de modifier les valeurs par défaut de ces variables, encore plus si le module que vous écrivez doit être utilisé sur d'autres magasins que le vôtre. Ils ne devraient être utilisés que si le module est pour votre propre magasin, et vous connaissez le groupe d'identification et de magasin de tous vos magasins.

Récupération de valeurs externes à partir du tableau de données `ps_configuration`

Vous n'êtes pas limité à vos propres variables: PrestaShop stocke tous ses propres paramètres de configuration dans la table de configuration de ps. Il y a littéralement des centaines de paramètres, et vous pouvez les accéder aussi facilement que vous auriez accès à vos propres. Par exemple:

- `Configuration::get( 'PS_LANG_DEFAULT' )`: récupère l'ID pour la langue par défaut.
- `Configuration::get( 'PS_TIMEZONE' )`: récupère le nom du fuseau horaire actuel, au format TZ standard (voir: http://en.wikipedia.org/wiki/List_of_tz_database_time_zones).
- `Configuration::get( 'PS_DISTANCE_UNIT' )`: récupère l'unité de distance par défaut ("km" pour des kilomètres, etc.).
- `Configuration::get( 'PS_SHOP_EMAIL' )`: récupère l'adresse e-mail principale du contact.
- `Configuration::get( 'PS_NB_DAYS_NEW_PRODUCT' )`: récupère le nombre de jours pendant lesquels un produit nouvellement ajouté est considéré comme "nouveau" par PrestaShop.

Plongez dans la `ps_configuration` table afin de découvrir de nombreux autres paramètres!

L'objet Shop

Une autre des lignes d'installation () est donc:

```
if ( Shop :: isFeatureActive () )
{
    Shop :: setContext ( Shop :: CONTEXT_ALL );
}
```

Comme indiqué précédemment, vérifiez que la fonctionnalité Multistore est activée et, dans l'affirmative, définissez le contexte actuel dans tous les magasins de cette installation de PrestaShop.

L'objet Shop vous aide à gérer la fonctionnalité multi-magasins. Nous ne plongerons pas dans les détails ici, mais nous présentons simplement les deux méthodes utilisées dans cet exemple de code:

- `Shop::isFeatureActive()`: Cela permet simplement de vérifier si la fonction multistore est active ou non, et si au moins deux magasins sont activés.
- `Shop::setContext(Shop::CONTEXT_ALL)`: Cela modifie le contexte afin d'appliquer les changements à venir à tous les magasins existants au lieu de seulement le magasin actuel.

Le contexte est expliqué dans plus de détails dans le chapitre "Utilisation de l'objet contextuel" de ce Guide du développeur.

Le fichier icône

Pour mettre fin à ce module de base, vous devez ajouter une icône qui sera affichée à côté du nom du module dans la liste des modules de back office. Dans le cas où votre module est conçu pour un service de premier plan, avoir visible le logo de ce service apporte confiance. Assurez-vous d'utiliser un logo déjà utilisé par l'un des modules natifs, ou sans l'autorisation du propriétaire du logo / service.

Le fichier icon doit respecter ces conditions:

- Il doit être placé sur le dossier principal du module.
- 32 * 32 image PNG.
- Nommé **logo.png**.
- Astuce: Il existe beaucoup de bibliothèques gratuites d'icônes 32 * 32 disponibles. Voici quelques-uns: <http://www.fatcow.com/free-icons> (très proche de FamFamFam) ou <http://www.iconarchive.com/show/danish-royalty-free-icons-by-jonas-rask.html> (Danish Royalty Free)

Installation du module

Maintenant que toutes les bases sont en place, rechargez les pages «Modules» du back office, dans la section «Fonctions du front office», vous devriez trouver votre module. Installez-le (ou réinitialisez-le s'il est déjà installé).

Pendant l'installation du module, PrestaShop crée automatiquement un petit **config.xml** fichier dans le dossier du module, qui stocke les informations de configuration. Vous devriez faire très attention lors de l'édition à la main.

À propos du fichier config.xml

Le **config.xml** fichier permet d'optimiser le chargement de la liste des modules dans le back office.

```
<? xml version = "1.0" encoding = "UTF-8"?>
<module>
  <nom> mymodule </ name>
  <displayName> <! [CDATA [Mon module]]> </ displayName>
  <version> <! [CDATA [1.0]]> </ version>
  <description> <! [CDATA [Description de mon module.]]> </ description>
  <author> <! [CDATA [Firstname Lastname]]> </ author>
  <onglet> <! [CDATA [front_office_features]]> </ tab>
  <confirmationUninstall> Êtes-vous sûr de vouloir désinstaller? </ confirmUninstall>
  <is_configurable> 0 </ is_configurable>
  <need_instance> 0 </ need_instance>
  <limited_countries> </ limited_countries>
</ module>
```

Quelques détails:

- **is_configurable** indique si le module a une page de configuration ou non.
- **need_instance** indique si une instance du module doit être créée lorsqu'elle est affichée dans la liste des modules. Cela peut être utile si le module doit effectuer des contrôles sur la configuration PrestaShop et afficher le message d'avertissement en conséquence.
- **limited_countries** est utilisé pour indiquer les pays auxquels le module est limité. Par exemple, si le module doit être limité en France et en Espagne, utilisez **<limited_countries>fr,es</limited_countries>**.

Ajout d'une page de configuration

Votre module peut obtenir un lien «Configure» dans la liste des modules de back office et permet donc à l'utilisateur de modifier certains paramètres. Ce lien "Configurer" apparaît avec l'ajout de la `getContent()` méthode à votre classe principale. C'est une méthode standard de PrestaShop: sa seule existence envoie un message au back office, en disant "il existe une page de configuration dans ce module, affiche le lien de configuration".

Mais avoir une `getContent()` méthode publique dans l'`MyModule` objet ne fait que créer le lien "Configure"; Il ne crée pas la page de configuration de nulle part. Nous allons expliquer comment créer un, où nous pourrions modifier le contenu de la `MYMODULE_NAME` variable que nous avons stockée dans le `ps_configuration` tableau de données.

La méthode getContent()

Tout d'abord, voici le code complet pour la méthode `getContent()`:

```
fonction publique getContent ()
{
    $ output = null;

    if (Tools :: isSubmit ('submit'. $ this-> name))
    {
        $ my_module_name = strval (Outils :: getValue ('MYMODULE_NAME'));
        si (! $ my_module_name
            || vide ($ my_module_name)
            || ! Valider :: isGenericName ($ my_module_name))
            $ output. = $ this-> displayError ($ this-> l ('Valeur de configuration invalide
            autre
            {
                Configuration :: updateValue ('MYMODULE_NAME', $ my_module_name);
                $ output. = $ this-> displayConfirmation ($ this-> l ('Paramètres mis à jour'));
            }
        }
        Retour $ output. $ this-> displayForm ();
    }
}
```

La `getContent()` méthode est la première à appeler lorsque la page de configuration est chargée. Par conséquent, nous l'utilisons pour mettre à jour toute valeur qui aurait pu être soumise par le formulaire que contient la page de configuration.

Voici une explication ligne par ligne:

- `Tools::isSubmit()` est une méthode spécifique à PrestaShop, qui vérifie si le formulaire indiqué a été validé. Dans ce cas, si le formulaire de configuration n'a pas encore été validé, le `if()` bloc entier est ignoré et PrestaShop n'utilisera que la dernière ligne, qui affiche la configuration avec les valeurs actuelles générées par la `displayForm()` méthode.
- `Tools::getValue()` est une méthode spécifique à PrestaShop, qui récupère le contenu du tableau POST ou GET afin d'obtenir la valeur de la variable spécifiée. Dans ce cas, nous récupérerons la valeur de la `MYMODULE_NAME` variable de formulaire, transformons sa valeur en une chaîne de texte à l'aide de la `strval()` méthode et la stockons dans la `$my_module_name` variable PHP.
- Nous vérifions ensuite l'existence d'un contenu réel `$my_module_name`, y compris l'utilisation de `Validate::isGenericName()`. L'objet Valider contient de nombreuses méthodes de validation de données, parmi lesquelles `isGenericName()` une méthode qui vous permet de conserver uniquement les chaînes valides des noms de PrestaShop - c'est-à-dire une chaîne qui ne contient pas de caractères spéciaux.
- Si l'une de ces vérifications échoue, la configuration s'ouvrira avec un message d'erreur indiquant que la validation du formulaire a échoué. Le , qui contient la version finale du code HTML qui fait la page de configuration, commence par un message d'erreur, créé à l'aide de la méthode de PrestaShop . Cette méthode renvoie le code HTML correct pour notre besoin, et puisque ce code est le premier à \$ output, cela signifie que la configuration s'ouvrira avec ce message.`$output` variable `displayError()`
- Si toutes ces vérifications réussissent, cela signifie que nous pouvons stocker la valeur dans notre base de données. Comme nous l'avons vu plus tôt dans ce tutoriel, l' `Configuration` objet a juste la méthode dont nous avons besoin: `updateValue()` stockerons la nouvelle valeur `MYMODULE_NAME` dans la table de données de configuration. À cette fin, nous ajoutons un message amical à l'utilisateur, indiquant que la valeur a effectivement été enregistrée: nous utilisons la

`displayConfirmation()` méthode PrestaShop pour ajouter ce message comme première donnée dans la variable `$output` - et donc, en haut de la page.

- Enfin, nous utilisons la `displayForm()` méthode personnalisée (que nous allons créer et expliquer dans la section suivante) afin d'ajouter du contenu `$output` (si le formulaire a été soumis ou non) et renvoyer ce contenu à la page. Notez que nous aurions pu inclure le code pour le `displayForm()` droit à l'intérieur `getContent()`, mais avons choisi de séparer les deux pour la lisibilité et la séparation des préoccupations.

Ce code de validation de formulaire n'est rien de nouveau pour les développeurs PHP, mais utilise certaines des méthodes PrestaShop que vous utiliserez très régulièrement.

Affichage du formulaire

Le formulaire de configuration lui-même est affiché avec la `displayForm()` méthode. Voici son code, que nous allons expliquer après le saut:

```
fonction publique displayForm ()
{
    // Obtenez une langue par défaut
    $ default_lang = (int) Configuration :: get ('PS_LANG_DEFAULT');

    // Ensemble des champs Init Fields
    $ fields_form [0] ['form'] = array (
        'légende' => tableau (
            'title' => $ this-> l ('Paramètres'),
        ),
        'input' => array (
            tableau (
                'type' => 'texte',
                'label' => $ this-> l ('Valeur de configuration'),
                'name' => 'MYMODULE_NAME',
                'taille' => 20,
                'required' => true
            )
        ),
        'submit' => array (
            'title' => $ this-> l ('Save'),
            'class' => 'btn btn-default pull-right'
        )
    );

    $ helper = new HelperForm ();

    // Module, token et currentIndex
    $ helper-> module = $ this;
    $ helper-> nom_controller = $ this-> nom;
    $ helper-> token = Outils :: getAdminTokenLite ('AdminModules');
    $ helper-> currentIndex = AdminController :: $ currentIndex. '& configure ='. $ this-> n;

    // La langue
    $ helper-> default_form_language = $ default_lang;
    $ helper-> allow_employee_form_lang = $ default_lang;

    // Titre et barre d'outils
    $ helper-> title = $ this-> displayName;
    $ helper-> show_toolbar = true; // false -> supprimer la barre d'outils
    $ helper-> toolbar_scroll = true; // oui -> La barre d'outils est toujours visible en haut
    $ helper-> submit_action = 'submit'. $ this-> name;
    $ helper-> toolbar_btn = array (
        'enregistrer' =>
            tableau (
                'desc' => $ this-> l ('Save'),
                'href' => AdminController :: $ currentIndex. '& configure ='. $ this-> name. '& :
                '& token ='. Tools :: getAdminTokenLite ('AdminModules'),
            ),
        'retour' => tableau (
            'href' => AdminController :: $ currentIndex. '& token ='. Tools :: getAdminTokenLite ('AdminModules'),
            'desc' => $ this-> l ('Retour à la liste')
        )
    );
}
```

```
);

// Charge la valeur actuelle
$ helper-> fields_value ['MYMODULE_NAME'] = Configuration :: get ('MYMODULE_NAME');

retour $ helper-> generateForm ($ fields_form);
}
```

Bien que cela puisse ressembler à un énorme bloc de code pour une seule valeur à changer, ce bloc utilise réellement la méthode de PrestaShop pour faciliter la création de formes, notamment l' `HelperForm` objet.

Plongée dans `displayForm ()`

Sortons cette méthode:

1. En utilisant la `Configuration::get()` méthode, nous récupérons la valeur de la langue actuellement choisie ("`PS_LANG_DEFAULT`"). Pour des raisons de sécurité, nous avons jeté la variable en entier en utilisant (`int`).
2. En prévision de la génération de la forme, nous devons créer un éventail de titres, de champs de textes et d'autres formes spécifiques.
À cette fin, nous créons la `$fields_form` variable, qui contiendra un tableau multidimensionnel. Chacun des tableaux qu'il comporte contient la description détaillée des tags que le formulaire doit contenir. À partir de cette variable, PrestaShop rendra le formulaire HTML tel qu'il est décrit.
Dans cet exemple, nous définissons trois balises (`<legend>`, `<input>` et `<submit>`) et leurs attributs à l'aide de tableaux. Le format est assez facile à obtenir: la légende et les tableaux de soumission contiennent simplement les attributs de chaque balise, tandis que l'entrée contient autant de balises nécessaires, chacune étant à son tour un tableau contenant les attributs nécessaires. Par exemple:

```
'input' => array (
    tableau (
        'type' => 'texte',
        'label' => $ this-> l ('Valeur de configuration'),
        'name' => 'MYMODULE_NAME',
        'taille' => 20,
        'required' => true
    ))
```

... génère les balises HTML suivantes:

```
< label > Valeur de configuration </ label > < div class = "margin-form" > < input id = "MYM
```

Comme vous pouvez le voir, PrestaShop est assez intelligent et génère tout le code nécessaire pour obtenir un formulaire utile. Notez que la valeur du tableau principal est effectivement récupérée plus tard dans le code de génération de formulaire.

3. Nous créons ensuite une instance de la `HelperForm` classe. Cette section du code est expliquée dans la section suivante de ce chapitre.
4. Une fois que les `HelperForm` paramètres sont tous en place, nous générons le formulaire en fonction du contenu de la `$fields_form` variable.

Utilisation de `HelperForm`

`HelperForm` est avec l'une des méthodes d'aide qui ont été ajoutés avec PrestaShop 1,5, `HelperOptions`, `HelperList`, `HelperView` et `HelperHelpAccess`. Ils vous permettent de générer des éléments HTML standard pour le back office ainsi que pour les pages de configuration du module. Vous pouvez obtenir plus d'informations sur les cours d'Aide dans le chapitre "Aideurs" de ce guide du développeur, avec une page dédiée `HelperForm`.

Voici notre exemple de code, comme un rappel:

```

$ helper = new HelperForm ();

// Module, Token et currentIndex
$ helper-> module = $ this;
$ helper-> nom_controller = $ this-> nom;
$ helper-> token = Outils :: getAdminTokenLite ('AdminModules');
$ helper-> currentIndex = AdminController :: $ currentIndex. '& configure ='. $ this-> name;

// La langue
$ helper-> default_form_language = $ default_lang;
$ helper-> allow_employee_form_lang = $ default_lang;

// titre et barre d'outils
$ helper-> title = $ this-> displayName;
$ helper-> show_toolbar = true; // false -> supprimer la barre d'outils
$ helper-> toolbar_scroll = true; // oui -> La barre d'outils est toujours visible en haut de
$ helper-> submit_action = 'submit'. $ this-> name;
$ helper-> toolbar_btn = array (
    'enregistrer' =>
        tableau (
            'desc' => $ this-> l ('Save'),
            'href' => AdminController :: $ currentIndex. '& configure ='. $ this-> name. '& save
            '& token ='. Tools :: getAdminTokenLite ('AdminModules'),
        ),
    'retour' => tableau (
        'href' => AdminController :: $ currentIndex. '& token ='. Tools :: getAdminTokenLite
        'desc' => $ this-> l ('Retour à la liste')
    )
);

// Charge la valeur actuelle
$ helper-> fields_value ['MYMODULE_NAME'] = Configuration :: get ('MYMODULE_NAME');

retour $ helper-> generateForm ($ fields_form);

```

Notre exemple utilise plusieurs **HelperForm** attributs: ils doivent être définis avant de générer le formulaire lui-même à partir de la `$fields_form` variable:

- `$helper->module`: nécessite l'instance du module qui utilisera le formulaire.
- `$helper->name_controller`: nécessite le nom du module.
- `$helper->token`: nécessite un jeton unique pour le module. `getAdminTokenLite()` nous aide à en générer un.
- `$helper->currentIndex`:
- `$helper->default_form_language`: requiert la langue par défaut pour le magasin.
- `$helper->allow_employee_form_lang`: requiert la langue par défaut pour le magasin.
- `$helper->title`: requiert le titre du formulaire.
- `$helper->show_toolbar`: nécessite une valeur booléenne - que la barre d'outils soit affichée ou non.
- `$helper->toolbar_scroll`: nécessite une valeur booléenne - que la barre d'outils soit toujours visible lors du défilement ou non.
- `$helper->submit_action`: nécessite l'attribut d'action pour la `<submit>` balise du formulaire .
- `$helper->toolbar_btn`: nécessite les boutons qui sont affichés dans la barre d'outils. Dans notre exemple, le bouton "Enregistrer" et le bouton "Retour".
- `$helper->fields_value[]`: c'est là que l'on peut définir la valeur de la balise nommée.

Enfin, après tout, on peut définir la `generateForm()` méthode, qui prendra soin de la mettre ensemble et, comme son nom l'indique, générer la forme que l'utilisateur utilisera pour configurer les paramètres du module.

Voici la restitution du formulaire tel qu'il est actuellement écrit - ce que vous pouvez voir par vous-même en cliquant sur le lien "Configurer" pour le module dans le back office:

Changez la valeur selon vos besoins, cliquez sur le bouton "Enregistrer", puis rechargez la page d'accueil: votre module est en effet mis à jour avec la nouvelle chaîne!

Affichage du contenu sur le front-office

En l'état, le module ne fait pas beaucoup. Pour afficher quelque chose sur le front office, nous devons ajouter un soutien pour quelques crochets. Cela se fait en mettant en œuvre les méthodes des crochets, et cela a été effectivement réalisé dans la `install()` méthode que nous avons écrite plus tôt, en utilisant la `registerHook()` méthode:

```
installation de la fonction publique ()
{
    si (Shop :: isFeatureActive ())
        Shop :: setContext (Shop :: CONTEXT_ALL);

    return parent :: install () &&
        $ this-> registerHook ('leftColumn') &&
        $ this-> registerHook ('header') &&
        Configuration :: updateValue ('MYMODULE_NAME', 'mon ami');
}
```

Comme vous pouvez le voir, nous le faisons pour que le module soit accroché aux crochets " `leftColumn`" et " `header`". En plus de cela, nous ajouterons un code pour le " `rightColumn`" crochet.

Le fait de joindre le code à un crochet nécessite une méthode spécifique pour chacun d'eux:

- `hookDisplayLeftColumn()`: accroche le code dans la colonne de gauche - dans notre cas, il récupérera le paramètre de module `MYMODULE_NAME` et affichera le fichier de modèle du module `mymodule.tpl`, qui doit être situé dans le `/views/templates/hook/dossier`.
- `hookDisplayRightColumn()`: tout simplement faire la même chose `hookDisplayLeftColumn()`, mais pour la colonne de droite.
- `hookDisplayHeader()`: Ajouter un lien vers le fichier CSS du module, `/css/mymodule.css`.

```
Fonction publique hookDisplayLeftColumn ($ params)
{
    $ this-> context-> smarty-> assign (
        tableau (
            'my_module_name' => Configuration :: get ('MYMODULE_NAME'),
            'my_module_link' => $ this-> context-> link-> getModuleLink ('mymodule', 'display')
        )
    );
    retourner $ this-> afficher (__ FILE__, 'mymodule.tpl');
}

gestion de la fonction publiqueDisplayRightColumn ($ params)
{
    retourne $ this-> hookDisplayLeftColumn ($ params);
}

Fonction publique hookDisplayHeader ()
{
    $ this-> context-> controller-> addCSS ($ this -> _ path.'css / mymodule.css ',' all ');
}
```

Nous utilisons `Context ($this->context)` pour changer une variable Smarty: la `assign()` méthode de Smarty permet de définir la variable de nom de modèle avec la valeur du `MYMODULE_NAME` paramètre stocké dans la table de base de données de configuration.

Le crochet d'en-tête ne fait pas partie de l'en-tête visuel, mais nous permet de mettre le code dans la `<head>` balise du fichier HTML généré. Ceci est très utile pour les fichiers JavaScript ou CSS. Pour ajouter un lien à notre fichier CSS dans la `<head>` balise de la page, nous utilisons la `addCSS()` méthode, qui génère la `<link>` balise correcte dans le fichier CSS indiqué dans les paramètres.

Enregistrez votre fichier et déjà vous pouvez accrocher le modèle de votre module dans le thème, le déplacer et le transplanter (même s'il n'y a pas de fichier modèle pour le moment): accédez à la page "Positions" dans le menu "Modules" dans le back office, puis cliquez sur le bouton "Transplanter un module" (en haut à droite de la page).

Dans la forme de transplantation:

- Trouvez «Mon module» dans la liste déroulante «Module».
- Choisissez "(displayLeftColumn) Blocs de colonne gauche" dans la liste déroulante "Accrocher dans".
- Cliquez sur "Enregistrer".

Il est inutile d'essayer d'attacher un module à un crochet pour lequel il n'a pas de méthode mise en œuvre.

La page "Positions" devrait être rechargée, avec le message suivant: "Module transplanté avec succès pour raccrocher" (ou peut-être "Ce module a déjà été transplanté sur ce crochet"). Toutes nos félicitations! Faites défiler la page "Positions", et vous devriez voir votre module parmi les autres modules dans la liste "Colonnes de gauche". Déplacez-le en haut de la liste en faisant glisser vers le bas la ligne du module.

Le module est maintenant attaché à la colonne de gauche ... mais sans aucun modèle à afficher, il ne fait rien d'utile: si vous rechargez la page d'accueil, la colonne de gauche affiche simplement un message où le module devrait être, en disant "Aucun modèle trouvé pour module mymodule".

Affichage du contenu

Maintenant que nous avons accès à la colonne de gauche, nous devrions afficher quelque chose ici pour le client à voir.

La partie visible du module est définie dans. `tpl` Fichiers placés dans des dossiers spécifiques:

- `/views/templates/front/`: fonctionnalités du front office.
- `/views/templates/admin/`: fonctionnalités de back office.
- `/views/templates/hook/`: fonctionnalités accrochées à un PrestaShop (peuvent donc être affichées sur le front office ou le back office).

Les fichiers de modèles peuvent avoir à peu près n'importe quel nom. Il n'y a qu'un seul fichier, il est bon de lui donner le même nom que le dossier et le fichier principal: `mymodule.tpl`.

Dans le cas de ce tutoriel, le module sera accroché à la colonne de gauche. Par conséquent, les fichiers TPL appelés à partir du crochet de la colonne doivent être placés `/views/templates/hook/` afin de fonctionner correctement.

Comme dit précédemment, le contenu à afficher dans le thème doivent être stockés dans des `.tpl` fichiers de modèles placés dans un dossier spécifique: `/views/templates/front/`. Les fichiers de modèles peuvent avoir à peu près n'importe quel nom. Il n'y a qu'un seul fichier, il est bon de lui donner le même nom que le dossier et le fichier principal: `mymodule.tpl`.

Nous allons créer le `mymodule.tpl` fichier, qui a été transmis en tant que paramètre de la `display()` méthode dans le code de notre module, dans la `hookDisplayHome()` méthode. Lorsque vous appelez un modèle à partir d'un crochet, PrestaShop recherche ce fichier modèle dans le `/views/templates/hook/` dossier (dans le dossier du module), que vous devez créer vous-même.

Dans PrestaShop 1.4, les fichiers de modèle du module devaient être placés à la racine du dossier du module.

Pour des raisons de compatibilité, les fichiers modèles peuvent encore résider dans le dossier racine dans PrestaShop 1.5 et 1.6, même si les sous-dossiers `/views/templates/` sont maintenant les emplacements recommandés. Si vous souhaitez que votre module fonctionne également dans PrestaShop 1.4, vous devez garder vos fichiers à la racine.

Voici notre fichier modèle, situé à `/views/templates/hook/mymodule.tpl`:

```
<!-- Bloquer mymodule -->
<div id = "mymodule_block_home" class = "block">
  <h4> Bienvenue! </ h4>
  <div class = "block_content">
    <p> Bonjour,
      {si isset ($ my_module_name) && $ my_module_name}
        {$ my_module_name}
      {autre}
        Monde
      {/si}
    </ p>
  <ul>
    <li> <a href="{ $ my_module_link}" title="Cliquez sur ce lien"> Cliquez-moi! </a> </ li>
  </ ul>
```

```
</ div>
</ div>
<! - / Block mymodule ->
```

Ce n'est qu'un code HTML régulier ... à l'exception de quelques appels Smarty:

- L' appel est une méthode spécifique à PrestaShop qui vous permet d'enregistrer la chaîne dans le panneau de traduction du module. Le paramètre `s` est la chaîne, alors que le paramètre doit contenir l'identifiant du module (dans le cas présent, "my-module"). Nous n'utilisons cette méthode qu'une seule fois pour des raisons de lisibilité, mais en pratique, elles devraient être utilisées sur toutes les chaînes de modèles. `{l s='xxx' mod='yyy'}mod`
- Les `{if}`, `{else}` et les `{/if}` déclarations sont conditionnels Smarty. Dans notre exemple, nous vérifions que la `$my_module_name` variable Smarty existe (en utilisant la `isset()` fonction PHP, considérée comme approuvée par Smarty) et qu'elle n'est pas vide. Si cela se passe bien, nous affichons le contenu de cette variable; sinon, nous affichons "World", afin d'avoir "Hello World".
- La `{$my_module_link}` variable dans l' `href` attribut du lien : il s'agit d'une variable Smarty que nous allons créer plus tard, qui indiquera le répertoire racine PrestaShop.

En plus de cela, nous allons créer un fichier CSS et l'enregistrer comme `/css/mymodule.css` dans le dossier du module (ou tout sous-dossier dans lequel vous souhaitez garder votre CSS):

```
div #mymodule_block_home p {
    font-size : 150 % ;
    police - le style : italic ;
}
```

Enregistrez le fichier modèle dans le `/views/templates/hook/` dossier du module et le fichier CSS dans le `/css/` dossier du module, rechargez la page d'accueil de votre boutique: le contenu du modèle devrait apparaître en haut de la colonne de gauche, juste en dessous du logo du magasin (si vous l'avez bien déplacé au haut du crochet "Colonne de gauche" pendant la partie de transplantation).

Comme vous pouvez le voir, le thème applique son propre CSS au modèle que nous avons ajouté:

- Notre `<h4>` titre devient l'en-tête du bloc, de la même manière que les autres titres de blocs.
- Notre bloc a le même style que les autres blocs de la page. `<div class="block_content">`

Ce n'est pas joli, mais ça fonctionne comme nous le voulons.

Désactivation du cache

Si vous avez suivi ce didacticiel sur la lettre et que vous ne voyez toujours rien qui apparaîtrait dans la colonne de gauche du thème, cela pourrait être dû au fait que PrestaShop a mis en cache les modèles précédents et vous sert toujours. Par conséquent, vous voyez la version originale du thème, sans vos modifications.

Smarty met en cache une version compilée de la page d'accueil, pour des raisons de performance. Ceci est extrêmement utile pour les sites de production, mais il est inutile pour les sites de tests, où vous pouvez charger la page avant très régulièrement afin de voir l'impact de vos modifications.

Lors de l'édition ou du débogage d'un thème sur un site de test, vous devez toujours désactiver le cache, afin de forcer Smarty à recompiler des modèles sur chaque chargement de page. À cette fin, accédez au menu "Paramètres avancés", sélectionnez la page "Performance", puis, dans la section "Smarty":

- Cache de modèles. Choisissez "Désactiver le cache".
- Cache. Désactivez-le.
- Console de débogage. Vous pouvez également ouvrir la console si vous souhaitez en savoir plus sur les composants internes de Smarty.

NE PAS désactiver le cache ou activer la console de débogage sur un site de production, car il ralentit tout en rien! Vous devez toujours effectuer tous vos tests dans un site de test, idéalement sur votre propre ordinateur plutôt qu'en ligne.

Incorporation d'un modèle dans le thème

Le lien que le module affiche ne mène nulle part pour le moment. Créez le **display.php** fichier auquel il cible, avec un contenu minimal, et placez-le dans le dossier racine du module.

Bienvenue sur cette page!

Cliquez sur le lien «Cliquez-moi!»: La page résultante est juste ce texte brut, sans rien du thème. Nous aimerions que ce texte soit intégré dans le thème, alors voyons comment faire exactement cela.

Comme vous vous attendez, nous devons créer un fichier modèle afin d'utiliser le style du thème. Créons le **display.tpl** fichier, qui contiendra la ligne de base "Bienvenue dans mon magasin!". Et sera appelé par **display.php**. Ce fichier display.php sera ré-écrit dans un contrôleur frontal afin d'intégrer correctement notre modèle de base dans l'en-tête, le pied de page, les colonnes, etc.

REMARQUE: vous devriez vous efforcer d'utiliser des noms explicites et reconnaissables pour vos fichiers de modèle, afin que vous puissiez les trouver rapidement dans le back office - ce qui est indispensable lors de l'utilisation de l'outil de traduction.

Voici nos deux fichiers:

```
display.php
<? php
classe mymoduleDisplayModuleFrontController extends ModuleFrontController
{
    Fonction publique initContent ()
    {
        parent :: initContent ();
        $ this-> setTemplate ('module: mymodule / views / templates / front / display.tpl');
    }
}

display.tpl
Bienvenue dans ma boutique!
```

Examinons **display.php**, notre premier contrôleur frontal de PrestaShop, stocké dans le dossier / controllers / avant du dossier principal du module:

```
Un contrôleur frontal doit être une classe qui étend la classe ModuleFrontController.
Ce contrôleur doit avoir une méthode: `initContent ()`, qui appelle la méthode `initContent
... qui appelle ensuite la méthode `setTemplate ()` avec notre fichier `display.tpl`.
```

setTemplate() est la méthode qui prendra soin d'intégrer notre modèle à une ligne dans une page pleine, avec l'en-tête, le pied de page et les barreaux appropriés.

Jusqu'à PrestaShop 1.4, les développeurs qui voulaient intégrer un fichier modèle dans le thème du site ont dû utiliser les appels `include ()` de PHP pour inclure chaque partie de la page. Voici le code équivalent pour display.php:

```
display.php
<? php
// Ce fichier doit être placé à la racine du dossier du module.
global smarty;
include ('../../ config / config.inc.php');
include ('../../ header.php');

$ smarty-> display (dirname (__ FILE __). '/ display.tpl');

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

Comme vous pouvez le voir, cela n'est plus nécessaire depuis PrestaShop 1.5: vous pouvez et utiliser un contrôleur frontal, et le contrôleur (Controller) et son modèle (View) doivent partager le même nom: **display.php** est lié à **display.tpl**.

Enregistrez les deux fichiers dans leurs dossiers respectifs, puis rechargez la page d'accueil de votre boutique, puis cliquez sur "Cliquez-moi!", Et voilà! Vous avez votre lien. Avec seulement quelques lignes, le résultat final est déjà beaucoup mieux, avec la ligne "Bienvenue" bien placée entre l'en-tête, le pied de page et les colonnes!

Ce n'est qu'une première étape, mais cela vous donne une idée de ce qui est possible si vous suivez les règles de la proposition.
Utilisation de Smarty

Smarty est un moteur de modèle PHP et est utilisé par le système de présélection de PrestaShop. C'est un projet gratuit et open source, hébergé à <http://www.smarty.net/>.

Il analyse les fichiers de modèle, recherche d'éléments dynamiques à remplacer par leurs équivalents contextuels, puis envoie le résultat généré au navigateur. Ces éléments dynamiques sont indiqués par des accolades: `{ ... }`. Les programmeurs peuvent créer de nouvelles variables et les utiliser dans les fichiers TPL; PrestaShop ajoute son propre ensemble de variables.

Par exemple, nous pouvons créer la variable `$ my_module_message` en PHP dans la `hookDisplayLeftColumn()` méthode et l'afficher par notre fichier modèle:

```
mymodule.php
Fonction publique hookDisplayLeftColumn ($ params)
{
    $ this-> context-> smarty-> assign (
        tableau (
            'my_module_name' => Configuration :: get ('MYMODULE_NAME'),
            'my_module_link' => $ this-> context-> link-> getModuleLink ('mymodule', 'display'),
            'my_module_message' => $ this-> l ('Ceci est un message texte simple') // N'oubliez pas de traduire
        )
    );

    retourner $ this-> afficher (__ FILE__, 'mymodule.tpl');
}
```

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

```
mymodule.tpl
{$ my_module_message}
```

PrestaShop ajoute son propre ensemble de variables. Par exemple, `{ $hook_left_column }` sera remplacé par le contenu de la colonne de gauche, c'est-à-dire le contenu de tous les modules qui ont été attachés au crochet de la colonne de gauche.

Toutes les variables Smarty sont globales. Vous devriez donc prêter attention à ne pas nommer votre propre variable avec le nom d'une variable Smarty existante, afin d'éviter de l'écraser. Il est conseillé d'éviter les noms trop simples, tels que `{ products }`, et ce préfixe avec le nom de votre module, ou même votre propre nom ou ses initiales, telles que: `{ $henryb_mymodule_products }`.

Voici une liste des variables Smarty qui sont communes à toutes les pages:

Dossier de fichiers	La description
img_ps_dir	URL pour le dossier d'image de PrestaShop.
img_cat_dir	URL pour le dossier d'images de catégories.
img_lang_dir	URL pour le dossier images images.
img_prod_dir	URL pour le dossier des images de produits.
img_manu_dir	URL pour le dossier d'images du fabricant.
img_sup_dir	URL pour le dossier des fournisseurs d'images.
img_ship_dir	URL pour le dossier d'images des transporteurs (expédition).
img_dir	URL pour le dossier d'images du thème.
css_dir	URL pour le dossier CSS du thème.
js_dir	URL pour le dossier JavaScript du thème.
tpl_dir	URL pour le dossier du thème actuel.
modules_dir	URL du dossier des modules.
mail_dir	URL pour le dossier des modèles de messagerie.
pic_dir	URL pour le dossier de téléchargement d'images.
lang_iso	Code ISO pour la langue actuelle.
viens de	URL pour l'origine du visiteur.

Dossier de fichiers	La description
shop_name	Nom du magasin.
cart_qties	Nombre de produits dans le panier.
Chariot	Le panier.
monnaies	Les différentes monnaies disponibles.
id_currency_cookie	ID de la devise actuelle.
devise	Objet de devise (monnaie actuellement utilisée).
biscuit	Cookie utilisateur.
les langues	Les différentes langues disponibles.
connecté	Indique si le visiteur est connecté à un compte client.
Nom de la page	Nom de la page.
clientName	Nom du client (si connecté).
priceDisplay	Méthode d'affichage des prix (avec ou sans taxes ...).
roundMode	Méthode d'arrondissement utilisée.
use_taxes	Indique si les taxes sont activées ou non.

Il existe de nombreux autres crochets contextuels. Si vous devez afficher toutes les variables Smarty de la page actuelle, ajoutez l'appel suivant:

```
{ debug }
```

Les commentaires sont basés sur l'astérisque:

```
{* Cette chaîne est commentée *}
```

```
{*
Cette chaîne est aussi trop!
*}
```

Contrairement aux commentaires HTML, le code Smarty commenté n'est pas présent dans le fichier de sortie final.

Traduction du module

Les chaînes de texte du module sont écrites en anglais, mais vous voudrez peut-être que les propriétaires de magasins français, espagnols ou polonais utilisent votre module aussi. Vous devez donc traduire ces chaînes dans ces langues, à la fois le front office et les chaînes de back office. Idéalement, vous devez traduire votre module dans toutes les langues installées sur votre boutique. Cela pourrait être une tâche fastidieuse, mais tout un système a été mis en place pour vous aider.

Bref, PrestaShop 1.7 implémente le mécanisme de traduction de Symfony, par l'utilisation de la `trans()` méthode, utilisé pour encapsuler les chaînes à traduire. Cette méthode est appliquée d'une manière différente selon le type de fichier.

Attention:

Ce mécanisme ne fonctionne pas avec les modules tiers. Vous devriez continuer à utiliser le [système de traduction hérité](#) et cela fonctionnera comme dans v1.6. Sinon, vous ne pourrez pas traduire vos modules, ce que nous voulons éviter.

Ces commentaires ne s'appliquent pas au développement du thème, car l'ensemble de la structure du thème a été redessiné en 1.7. Les thèmes spécifiques à 1.7 devraient utiliser le nouveau système de traduction. Juste pas les modules :(

Le processus de préparation des chaînes de texte pour la traduction s'appelle internationalisation, ou i18n.

Internationalisation de chaînes dans les fichiers Smarty (.tpl)

Les chaînes dans les fichiers TPL devront être transformées en contenu dynamique en utilisant l'`{l}` appel de fonction, que Smarty remplacera par la traduction pour la langue choisie.

PrestaShop 1.6 utilisait pour exiger le `mod` paramètre pour le contexte. PrestaShop 1.7 nécessite maintenant que ce paramètre soit "d" et qu'il utilise le même domaine que toutes les autres chaînes du module.

Dans notre module exemple, le `mymodule.tpl` fichier ...

```

mymodule.tpl (partiel)
<li>
  <a href="{ $base_dir }modules/mymodule/mymodule_page.php" title="Cliquez sur ce lien"> Cliquez
</ li>
<!-- Bloquer mymodule -->
<div id = "mymodule_block_left" class = "block">
  <h4> {ls = 'Bienvenue!' d = 'Modules.MyMode'} </ h4>
  <div class = "block_content">
    <p> Bonjour,
      {si isset ( $ my_module_name ) && $ my_module_name }
        { $ my_module_name }
      {autre}
        Monde
      {/si}
    </ p>
    <ul>
      <li> <a href="{ $my_module_link }" title="Cliquez sur ce lien"> Cliquez-moi! </a> </ li>
    </ ul>
  </ div>
</ div>
<!-- / Block mymodule -->

```

...devient:

```

mymodule.tpl (partiel)
<li>
  <a href="{ $base_dir }modules/mymodule/mymodule_page.php" title="{ls='Cliquez sur ce lien 'm
</ li>
<!-- Bloquer mymodule -->
<div id = "mymodule_block_left" class = "block">
  <h4> {ls = 'Bienvenue!' d = 'Modules.MyMode'} </ h4>
  <div class = "block_content">

```

```

<p>
    {if! isset ($ my_module_name) || ! $ my_module_name}
        {capture name = 'my_module_tempvar'} {ls = 'World' d = 'Modules.MyModule'} {/ capture}
        {assign var = 'my_module_name' value = $ smarty.capture.my_module_tempvar}
    {/si}
    {ls = 'Bonjour% 1 $ s!' sprintf = $ my_module_name d = 'Modules.MyModule'}
</ p>
<ul>
    <li> <a href="{ $ my_module_link}" title="{ls='Cliquez sur ce lien 'd='Modules.MyModule' }
</ ul>
</ div>
</ div>
<!-- / Block mymodule -->

```

... et le `display.tpl` fichier:

```

display.tpl
Bienvenue sur cette page!

```

...devient:

```

afficher . tpl
{ l s = 'Bienvenue sur cette page!' d = 'Modules.MyModule' }

```

Notez que nous utilisons toujours le `d` paramètre. Ceci est utilisé par PrestaShop pour affirmer à quel module appartient la chaîne. L'outil de traduction l'exige afin de faire correspondre la chaîne à traduire avec sa traduction. Ce paramètre est obligatoire pour la traduction de modules.

Stratégies d'internationalisation dans les fichiers Twig (.twig)

Pour être écrit.

Génération de vos fichiers de traduction

Pour être écrit.

Traduire les chaînes de votre module

Les chaînes sont délimitées avec des guillemets simples. Si une chaîne contient des guillemets simples, elles devraient être échappées à l'aide d'une barre oblique inverse (\).

De cette façon, les chaînes peuvent être traduites directement dans PrestaShop:

- Accédez à la page "Traductions" sous le menu "Localisation"
- Dans le menu déroulant "Modify translations", choisissez "Modules de modules installés",
- Choisissez la langue dans laquelle vous voulez traduire le module. La langue de destination doit déjà être installée pour permettre la traduction.
- Cliquez sur le bouton "Modifier".

La page qui charge affiche toutes les chaînes pour tous les modules actuellement installés. Les modules qui ont toutes leurs chaînes déjà traduites ont leur champ clos, alors que si au moins une chaîne manque dans la traduction d'un module, son champ est étendu. Pour traduire les chaînes de votre module (celles qui ont été "marquées" à l'aide de la `l()` méthode), recherchez simplement votre module dans la liste (utilisez la recherche dans la page du navigateur) et remplissez les champs vides.

Une fois que toutes les chaînes de votre module sont correctement traduites, cliquez sur le bouton «Enregistrer et rester» ou sur le bouton «Enregistrer» au bas de votre liste de chaînes.

PrestaShop enregistre ensuite les traductions dans un nouveau fichier, nommé en utilisant le `languageCode.php` format (par exemple `/mymodule/fr.php`). Le fichier de traduction ressemble ainsi:

```
fr.php
```

```
<? php
global $_MODULE;
$_MODULE = array ();
$_MODULE ['<{mymodule} prestashop> mymodule_2ddddc2a736e4128ce1cdfd22b041e7f'] = 'Mon module';
$_MODULE ['<{mymodule} prestashop> mymodule_d6968577f69f08c93c209bd8b6b3d4d5'] = 'Description';
$_MODULE ['<{mymodule} prestashop> mymodule_533937acf0e84c92e787614bbb16a7a0'] = 'Êtes-vous';
$_MODULE ['<{mymodule} prestashop> mymodule_0f40e8817b005044250943f57a21c5e7'] = 'Aucun nom';
$_MODULE ['<{mymodule} prestashop> mymodule_fe5d926454b6a8144efce13a44d019ba'] = 'Valeur de';
$_MODULE ['<{mymodule} prestashop> mymodule_c888438d14855d7d96a2724ee9c306bd'] = 'Réglages';
$_MODULE ['<{mymodule} prestashop> mymodule_f4f70727dc34561dfde1a3c529b6205c'] = 'Réglages';
$_MODULE ['<{mymodule} prestashop> mymodule_2f6e771db304264c8104cb7534bb80cd'] = 'Valeur de';
$_MODULE ['<{mymodule} prestashop> mymodule_c9cc8cce247e49bae79f15173ce97354'] = 'Enregistré';
$_MODULE ['<{mymodule} prestashop> mymodule_630f6dc397fe74e52d5189e2c80f282b'] = 'Retour à';
$_MODULE ['<{mymodule} prestashop> display_86e88cbccafa83831b4c6685501c6e58'] = 'Bienvenue';
$_MODULE ['<{mymodule} prestashop> mymodule_9a843f20677a52ca79af903123147af0'] = 'Bienvenue';
$_MODULE ['<{mymodule} prestashop> mymodule_f5a7924e621e84c9280a9a27e1bcb7f6'] = 'Monde';
$_MODULE ['<{mymodule} prestashop> mymodule_3af204e311ba60e6556822eac1437208'] = 'Bonjour%';
$_MODULE ['<{mymodule} prestashop> mymodule_c66b10fbf9cb6526d0f7d7a602a09b75'] = 'Cliquez sur';
$_MODULE ['<{mymodule} prestashop> mymodule_f42c5e677c97b2167e7e6b1e0028ec6d'] = 'Cliquez-moi';
```

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

Maintenant que nous avons une traduction en français, nous pouvons cliquer sur le drapeau français au front office et obtenir le résultat attendu: les cordes du module sont maintenant en français.

Ils sont également traduits en français lorsque le back office est en français.

Traduire le code complexe

Comme on peut le voir, la base de la traduction du fichier modèle est de les joindre au . Les changements dans et dans les textes de lien et le titre » sont donc faciles à comprendre. Mais a ajouté un bloc de code plus délicat pour la chaîne "Hello World!": Une

```
clause if / else / then et une variable de texte. Explorez ce code:
{ls = 'The string'
mod='name_of_the_module'}display.tplmymodule.tpl
```

Voici le code original:

```
Bonjour,
{si isset ($ my_module_name) && $ my_module_name}
    {$ my_module_name}
{autre}
    Monde
{/si}
!
```

Comme vous pouvez le voir, nous devons obtenir la chaîne "Hello World" traduisible, mais aussi pour répondre au fait qu'il existe une variable. Comme expliqué dans le chapitre "Traductions dans PrestaShop 1.5", les variables doivent être marquées à l'aide de `sprintf()` marqueurs, tels que `%sou %1$s`.

Faire des mots traduisants "Hello% s!" Facilement: il suffit d'utiliser ce code:

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

Mais dans notre cas, nous devons également nous assurer que le% s est remplacé par "World" dans le cas où la valeur "my_module_name" n'existe pas ... et nous devons également traduire "World". Cela peut être réalisé en utilisant la `{capture}` fonction Smarty , qui collecte la sortie du modèle entre les variables dans une variable au lieu d'afficher, afin que nous puissions l'utiliser plus tard. Nous allons l'utiliser pour remplacer la variable par le "Monde" traduit si la variable est vide ou absente, en utilisant une variable temporaire. Voici le code final:

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

Module de paiement

PrestaShop 1.7 présente une nouvelle API de paiement. Ci-dessous, vous trouverez des informations sur la façon dont cela fonctionne et aussi, comment migrer votre module de PrestaShop 1.6 (et plus tôt) vers PrestaShop 1.7.

La raison principale pour laquelle le changement était nécessaire est du côté client: il n'y a **qu'un seul** bouton pour valider la commande, pas plus par module de paiement.

Exigences

Pour créer un module de paiement pour PrestaShop 1.7, vous devrez respecter certains éléments:

- Votre classe devra étendre *PaymentModule* .
- Vous devrez déclarer l'espace de noms suivant.

```
utilisez PrestaShop \ PrestaShop \ Core \ Payment \ PaymentOption;
```

- Vous devrez enregistrer les deux méthodes suivantes: *hookPaymentOptions ()* & *hookPaymentReturn ()* et enregistrez ces crochets.
- Vous ne devez pas avoir de bouton de soumission dans le code HTML de votre module. Il sera automatiquement généré par PrestaShop.

Dans la méthode *hookPaymentOptions ()* , vous devez renvoyer un tableau de *PaymentOption_* .

Comment générer votre PaymentOption

PaymentOption

Voici une liste des variables de classe *PaymentOption* . Ils ont tous un getter et un setter et sont accessibles dans le tableau envoyé au modèle de front office.

- **\$ callToActionText:** le texte affiché comme nom de la méthode de paiement.
- **\$ additionalInformation:** informations supplémentaires à afficher au client. Ceci est un code HTML gratuit, et peut être utilisé par des modules tels que bankwire à afficher à quel compte le transfert bancaire doit être effectué.
- **\$ logo:** L'URL d'une image à afficher dans le widget de sélection des options de paiement.
- **\$ action:** l'URL à laquelle la demande de traitement du paiement doit être effectuée.
- **\$ input:** Un ensemble associatif de paramètres supplémentaires à utiliser lors de l'envoi de la demande à \$ action.
- **\$ forme:** le HTML personnalisé à afficher comme un formulaire pour entrer les informations de carte de crédit.
- **\$ iframe:** le HTML personnalisé contenant un iframe avec le widget de confirmation de paiement pour les modules comme ATOS.
- **\$ moduleName:** le nom du module.
- **\$ binary:** Un boolean à définir si le formulaire de module a été généré par des binaires et contient un bouton de soumission. Il est nécessaire d'adapter le comportement.

Types de PaymentOption

Vous pouvez trouver un exemple de module illustrant les quatre cas identifiés de modules de paiement sur [GitHub](#) .

Nous avons identifié quatre cas de module de paiement dans lesquels vous pouvez être:

Hors ligne:

C'est le cas le plus simple où vous pourriez être (p. Ex.: Bankwire, Check)

C'est une URL simple à appeler, puis diverses informations sont affichées au client.

Les variables minimales à définir sont *\$ callToActionText* et *\$ action* . Vous pouvez vérifier la méthode *getOfflinePaymentOption ()* de *paymentexample_* pour avoir un exemple.

Externe:

C'est une URL simple à appeler, puis le paiement est directement traité sur le site Web du fournisseur de services de paiement (p. Ex.: PayPal, Paybox).

Les variables minimales à définir sont `$ callToActionText` et `$ action` . Le formulaire sera envoyé par POST et vous pouvez ajouter des entrées cachées, comme un jeton, en utilisant la méthode `setInput ()` . Vous pouvez vérifier la méthode `getExternalPaymentOption ()` de `paymentexample_` pour avoir un exemple.

Embedded:

Vous écrivez votre numéro de carte de crédit et toutes les données requises directement sur le site du marchand (p. Ex., Stripe).

Les variables minimales à définir sont `$ callToActionText` et `$ form` . Vous pouvez vérifier la méthode `getEmbeddedPaymentOption ()` de `paymentexample_` pour avoir un exemple.

Iframe:

Le formulaire de paiement est affiché sur le site du marchand, mais à l'intérieur d'un iFrame.

Les variables minimales à définir sont `$ callToActionText` et `$ additionalInformation` . Vous pouvez vérifier la méthode `getIframePaymentOption ()` de `paymentexample_` pour avoir un exemple.

Migration de 1,6 à 1,7

How-To

Vous devez modifier le *gain de paiement* dans lequel votre module est accroché par `paymentOption` . **Ce n'est plus un crochet d'affichage** , donc vous ne devez pas utiliser la méthode `$ this-> display ()` pour récupérer un modèle, mais utilisez la méthode `$ this-> context-> smarty-> fetch ()` à la place.

Ensuite, implémentez la fonction `hookPaymentOptions ()` pour renvoyer un tableau de `PaymentOption_` .

Ensuite, vous devez identifier le type de votre module de paiement pour savoir quelles variables sont obligatoires.

Que faire si je ne supprime pas le bouton de bonification?

Comme vous pouvez le lire ci-dessus, vous ne devez pas avoir de bouton de soumission dans le code HTML de votre module, car PrestaShop le générera automatiquement. Si vous ne pouvez pas supprimer le bouton de soumission du formulaire pour certaines raisons (par exemple: le formulaire est généré par des binaires), nous avons mis en place une autre façon de rendre compatible votre module PrestaShp 1.7. Mais, notez que ce n'est **pas** la façon recommandée de le faire.

Pour ce faire, vous devrez implémenter un crochet supplémentaire: `displayPaymentByBinaries` . Il est fait pour afficher le formulaire de paiement, et il remplacera le bouton de paiement unique à la caisse.

Vous devrez également définir la variable `$ binary` sur `true` . Il adaptera le comportement pour masquer le bouton de paiement et le remplacer par le formulaire lorsque l'option de paiement est sélectionnée.

Crochets

Les params passés aux crochets suivants ont été modifiés:

- `hookPaymentReturn`
- `hookDisplayOrderConfirmation`

AVANT

Clé	Valeur
le total à payer	résultat de <code>\$ order-> getOrdersTotalPaid ()</code>
devise	signe de devise (chaîne)
Clé <code>currencyObj</code>	Valeur La monnaie chargée (classe de devise)
<code>objOrder</code>	L'objet de commande actuel (Classe de commande)

APRÈS

Clé	Valeur
commande	L'objet de commande actuel (Classe de commande)

Tout peut être récupéré, par exemple:

```
$ currency = new Currency ($ params ['order'] -> id_currency);  
$ total_to_pay = $ params ['order'] -> getOrdersTotalPaid ();
```

Exemple de vie réelle

Consultez le module bankwire pour un exemple de la façon dont nous avons mis à jour un module 1.6 simple vers 1.7:

<https://github.com/PrestaShop/bankwire/pull/18>

Migrant de 1,6

BACKWARD COMPAT 'BROKEN':

- Le module de paiement a changé, vérifiez doc.

ASSURER:

- Nous avons changé toutes les variables passées à smarty, assurez-vous que ce que vous utilisez est toujours affecté
- Le lien \$ est obsolète (même s'il est encore transmis à smarty, il sera supprimé à un certain point), utilisez {url} helper
- assurez-vous d'utiliser la nouvelle façon d'appeler les modèles: *fetch (module: modulename / views / template.tpl)*
- assurez-vous de ne pas ajouter vos actifs avec un chemin codé en mode DisplayHeader (pas de changement mais c'est mauvais)
- Smarty est automatiquement échappé, donc: 1. Supprimez tous | *escape: 'html': 'UTF-8'* (POUR LES MODÈLES AVANT UNIQUEMENT) 2. Ajoutez le *filtre* sans *fil* si vous imprimez html
- Ne comptez que des variables documentées! Toute autre variable que je viens d'un module ou simplement une partie restante et peut être supprimée à tout moment.
- Si vous utilisez le package *backward_compatibility*, vérifiez que la version PrestaShop est 1.4 avant de l'importer. Avec PrestaShop 1.7, vous obtiendrez une erreur fatale

Remarque: de nombreuses bibliothèques externes ont été mises à jour. Voici une liste de certains documents supplémentaires externes à ce sujet:

- <https://github.com/yellowtree/wp-geoip-detect/wiki/How-to-migrate-from-v1-to-v2>

Comment rendre un module compatible avec 1.6 ET 1.7

- Utilisez le système de traduction hérité (\$ this-> l)
- Utilisez 2 modèles différents selon la version
- Implémentez *getWidgetVariables* et *renderWidget* mais ne met pas en œuvre explicitement l'interface php.
- Créez toute méthode de crochet spécifique comme *hookDisplayHeader* et appelez le *renduWidget* avec les paramètres corrects