[Astuce] Ne pas coder “en dur” la variable id_lang dans les modules Prestashop

En effet, entre Prestashop 1.4 et 1.5, les valeurs par défaut de la table ps_lang ont changé. Donc la valeur de id_lang pour le français est passé de 2 à 4.

Evitez donc de la coder en dur dans vos modules (c’est-à-dire mettre la valeur numérique dans vos requêtes, du style “WHERE id_lang=2”) et privilégiez plutôt :

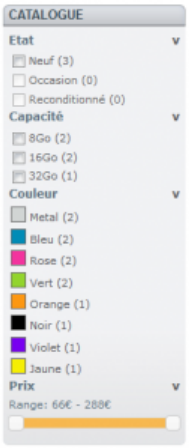
$cookie->id_lang pour Prestashop 1.4

Context::getContext()->language->id pour Prestashop 1.5

Utiliser la navigation à facettes de Prestashop

La navigation à facettes, apparue avec la version 1.4 de Prestashop (mais réellement utilisable à partir de la version 1.4.5), permet d’une manière assez simple de définir des critères pour filtrer les résultats d’une catégorie (comme on le retrouve sur Pixmania ou d’autres sites de e-commerce).Elle fonctionne de la manière suivante : vous allez déterminer des modèles (c’est à dire un ensemble de

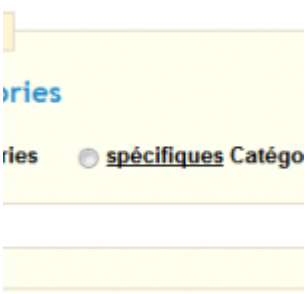
critères, comme les caractéristiques (capacité, etc), les marques, l'état du stock, etc).Ensuite vous allez pouvoir associer ces modèles à des catégories, par exemple le modèle "pour les appareils photo" aux catégories comprenant les appareils photo.



Chaque modèle pourra être utilisé pour autant de catégories que vous le souhaitez.

Pour mettre en place la navigation à facettes sur une catégorie, suivez ces étapes :

- Dans l'administration de votre boutique, allez dans Allez dans **Modules / Bloc navigation à facettes**
- Allez dans **Construisez vos propres modèles de filtre** et cochez **spécifiques Catégories (0 sélectionnée(s))** :

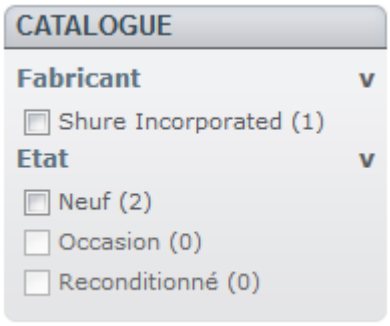


- Dans le bloc qui s'affiche alors, sélectionnez les catégories sur lesquelles vous voulez appliquer ce modèle et cliquez sur **Sauvegarder la sélection** :

- Dans la liste des critères qui s'affiche alors, vous allez pouvoir en sélectionner certains, en les cochant :

- Donnez ensuite un nom à votre sélection de critères et cliquez sur **Sauvegarder ce modèle de filtres**

Si on prend l'exemple ci-dessus, cela donnera coté boutique, seulement pour la catégorie "Accessoires", un bloc de navigation à facettes comme celui-là :



Note importante : les critères disponibles sur la droite sont dépendants de la catégorie que vous avez choisi.

Dans l'exemple cité au dessus, aucun produit n'a de "caractéristiques" (au sens Prestashop). Si à l'inverse je sélectionne la catégorie "iPods" qui contient, elle, des produits ayant des "caractéristiques", les choix à notre disposition seront différents :

Rajouter un nouveau hook dans Prestashop 1.4 grace à l'override

Prestashop, dans sa dernière version (de mars 2011), propose une vraie amélioration (parmi beaucoup d'autres) : l'override.

Avant de rentrer dans les détails de ce système, rappelons comment est architecturé Prestashop :

- Les classes, dans **/classes**, définissent le comportement des fonctionnalités principales (Cart, Order, Customer, etc)
- Les modules, dans **/modules**, permettent d'étendre les possibilités de base (newsletter, alertes email, paiement, etc)

Parallèlement, il existe à de nombreux endroits dans Prestashop des hooks ("crochets") qui sont des points dans le code sur lesquels les modules peuvent se brancher (dans les classes, ou même dans les fichiers à la racine, qui gèrent le

panier, le compte-client, etc).

Une bonne manière de comprendre le fonctionnement de ces “hooks” est de les comparer au déroulement d’une vie : vous pouvez intervenir à des moments stratégiques de celle-ci, pour modifier ou non le comportement d’une personne. C’est un peu pareil avec les hooks : ils permettent aux modules de déclencher des actions à des moments-clés du fonctionnement de Prestashop (à l’affichage de l’en-tête, mais aussi au passage d’une commande ou lors de son annulation, etc). Autre exemple : un développeur peut décider d’utiliser le hook “panier” et d’afficher à cet emplacement du contenu (par exemple la météo ou des produits associés).

Certains hooks sont “invisibles” et sont plus des hooks de calcul ou de traitement (génération de PDF, envoi de mail).

Le problème avec ces hooks est qu’ils sont pré-existants dans l’application : en clair, il peut arriver qu’il en manque un précisément là où il serait nécessaire.

Jusqu’à Prestashop 1.3.7, le seul moyen était de le rajouter à la main dans le code, mais *avec le risque qu’à la mise à jour suivante, ces modifications disparaîtraient*.

Depuis Prestashop 1.4, il existe un dispositif qui permet de résoudre ce problème : **l’override**.

Derrière cet anglicisme se cache tout simplement la possibilité de “surcharger” Prestashop, c’est à dire de redéfinir ce que fait l’application, et sans craindre de perdre ces changements à la prochaine évolution.

En effet, ces “surcharges” se situent dans un répertoire **/override**, vide par défaut dans les versions téléchargeables de l’outil, donc qui ne sera pas écrasé par une mise à jour.

Prenons un exemple pratique, pour voir comment tout cela se

décompose.

Pitch : *On désire rajouter du contenu avant le footer, dans une zone qu'on appellera "surfooter", indépendante du bas de page*.

– 1ère étape : Créer le hook dans la base de données

En effet, pour que Prestashop puisse déclencher une action sur un hook, il faut que celui-ci soit référencé dans la liste des choix possibles.

On va ici utiliser **PHPMyAdmin** pour l'insérer. Rendez-vous dans la table **ps_hook** (ou toute table finissant par "_hook", selon le préfixe que vous avez utilisé). Cliquez sur **Insérer**, et complétez comme suit l'écran qui apparaît :

Et cliquez sur **Exécuter**.

Pour les fans du SQL, la commande suivante est équivalente :

```
INSERT INTO ps_hook (name,title,position,live_edit) VALUES ('surfooter', 'Hook surFooter', '1', '0')
```

– 2ème étape : Positionner le hook dans le code de l'application

Dans l'exemple qu'on a choisi, le "surfooter" doit se déclencher juste avant le "footer". Comment se déclenche le hook "footer" ? Regardons dans le fichier **footer.php** (logique, non?) :

```
$controller = new FrontController();
$controller->displayFooter();
```

Depuis la 1.4, on utilise dans Prestashop des contrôleurs, réunis dans un dossier **/controllers**, qui contiennent les fonctions utilisées dans les différentes pages.

Regardons le code de la fonction displayFooter dans **/classes/FrontController.php** :

```
public function displayFooter()
	{
		global $cookie;
		if (!self::$initialized)
			$this->init();

		self::$smarty->assign(array(
					'HOOK_RIGHT_COLUMN' => Module::hookExec('rightColumn', array('cart' => self::$cart)),
					'HOOK_FOOTER' => Module::hookExec('footer'),
					'content_only' => (int)(Tools::getValue('content_only'))));
self::$smarty->display(_PS_THEME_DIR_.'footer.tpl');
		//live edit
		 if (Tools::isSubmit('live_edit') AND $ad = Tools::getValue('ad') AND (Tools::getValue('liveToken') == sha1(Tools::getValue('ad')._COOKIE_KEY_)))
		{
			 self::$smarty->assign(array('ad' => $ad, 'live_edit' => true));
self::$smarty->display(_PS_ALL_THEMES_DIR_.'live_edit.tpl');
		}
		else
			Tools::displayError();
	}
```

La ligne **HOOK_FOOTER' => Module::hookExec('footer')** permet de déclencher "les fonctions hookfooter situés dans les modules installés sur la boutique". Parallèlement, on retrouve dans le template un bloc **{$HOOK_FOOTER}** qui affichera là ou il est inséré le contenu généré par les fonctions qu'on vient évoquer.

Pour notre “surfooter”, il suffirait donc de rajouter :

```
HOOK_SURFOOTER' => Module::hookExec('surfooter'),
```

Mais (il y a un “mais”), comme on l’a précisé au début de l’article, ces fichiers du coeur de Prestashop seront écrasés à la prochaine mise à jour. On va donc “dupliquer” cette fonction, et créer un fichier **/override/classes/FrontController.php**, contenant :

```
//La classe d'origine située dans /classes/FrontController.php s'appelle FrontControllerCore, on utilise donc ici FrontController
class FrontController extends FrontControllerCore
{
         //Copie de la fonction displayFooter située dans /classes/FrontController.php, mais avec le "surfooter" en +
        //C'est la copie qui prendra la main sur la fonction d'origine appelée dans /footer.php
    public function displayFooter()
        {
                global $cookie;
                if (!self::$initialized)
                        $this->init();

                self::$smarty->assign(array(
                                        'HOOK_RIGHT_COLUMN' => Module::hookExec('rightColumn', array('cart' => self::$cart)),
                                //On rajoute l'appel à notre nouveau hook
                                        'HOOK_SURFOOTER' => Module::hookExec('surfooter'),
                                        'HOOK_FOOTER' => Module::hookExec('footer'),
                                        'content_only' => (int)(Tools::getValue('content_only'))));
self::$smarty->display(_PS_THEME_DIR_.'footer.tpl');
                //live edit
                  if (Tools::isSubmit('live_edit') AND $ad = Tools::getValue('ad') AND (Tools::getValue('liveToken') == sha1(Tools::getValue('ad')._COOKIE_KEY_)))
                {
```

```
                        self::$smarty->assign(array('ad' =>
$ad, 'live_edit' => true));
self::$smarty->display(_PS_ALL_THEMES_DIR_.'live_edit.tpl');
                }
                else
                        Tools::displayError();
        }
}
```

– 3ème étape : “greffer” un module sur notre hook

Que l’on souhaite être directement connecté au “surfooter”, ou qu’on souhaite les relier ensuite par “Greffer un module”, la procédure est la même : il faut déclarer au début du fichier du module que l’on souhaite s’y connecter.

Prenons l’exemple de “Bloc newsletter”, (**/modules/blocknewsletter/blocknewsletter.php**) et de sa fonction **install()** (présente dans toutes les extensions Prestashop) :

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

On voit qu’est utilisée plusieurs fois la fonction **registerHook**, qui permet de se “référencer” sur un hook précis. On va donc modifier cette ligne ainsi :

```
if      (parent::install()      ==      false      OR
$this->registerHook('leftColumn') == false
OR   $this->registerHook('header')   ==   false    OR
$this->registerHook('surfooter') == false) {
        return false;
}
```

Désormais, à la prochaine installation de ce module (donc par exemple si on le désinstalle/réinstalle), il se branchera sur “surfooter” et cherchera à éxécuter sa propre fonction **hooksurfooter()**, qu’on va créer pour l’occasion (toujours dans

le même fichier **/modules/blocknewsletter/blocknewsletter.php**) :

```
function hooksurfooter($params)
        {
                    //Mettre ici du code à exécuter dans le surfooter
        }
```

Il faudra bien sur rajouter le bout de code nécessaire dans notre template, par exemple:

```
{$HOOK_SURFOOTER}
```

(merci à Jérémy Kleinclaus pour la remarque à ce sujet là)
Bien sur, comme toute modification importante sur un template et sur la boutique, il peut être nécessaire de vider le cache Prestashop si les changements n'apparaissent pas.

On peut également utiliser ce procédé pour modifier le fonctionnement d'une fonction du core de Prestashop, sans nécessairement vouloir rajouter un hook, ou "surcharger" un contrôleur, en utilisant le répertoire **/override/controllers**.