# Adapter votre module à Bootstrap

## Adapter votre module à Bootstrap

La version 1.6 de PrestaShop offre un tout nouveau design au thème par défaut et au logiciel lui-même. Ces designs sont techniquement basés sur le framework CSS Bootstrap 3 (http://getbootstrap.com/), qui permet aux designers et aux développeurs de se baser sur ses outils et modèles pour créer de superbes designs responsifs.

En tant que développeur de module, il est de votre intérêt de mettre à jour votre module pour qu'il prenne en compte Bootstrap, afin de s'intégrer au nouveau design.

### Tout se passe dans les helpers

En soi, ce n'est pas bien compliqué : le plus gros du travail est pris en charger par les méthodes Helper de PrestaShop, qui ont été mises à jour pour utiliser Bootstrap de la bonne manière. De fait, le plus dur sera pour vous de modifier le code d'interface de votre module pour qu'il abandonne l'ancienne manière, ou vous définissiez les formulaires directement en HTML, et adopte la nouvelle manière introduite lors de la version 1.5 de PrestaShop, et qui fait que les méthodes Helper font tout le travail.

### Avant / après

Voici un exemple de la manière dont votre module peut exploiter le framework Bootstrap, par le biais des méthodes Helper. Cet exemple est tiré du module Bloc Panier, et se concentre sur deux méthodes : `getContent()` (que PrestaShop appelle afin d'afficher la page de configuration du module), et `displayForm()` (qui remplace `renderForm()`).

Vous pouvez voir la différence directement sur Github :

- Ancien module Bloc Panier : https://github.com/PrestaShop/PrestaShop/blob/62ff976d69f4f5efd3413227f20bed429705e7b7/modules/blockcart/blockcart.php
- Module Bloc Panier bootstrappé : https://github.com/PrestaShop/PrestaShop/blob/8144935d764d39d9ed809a1d16c8f452dd9f5591/modules/blockcart/blockcart.php
- La modification du code qui a tout mis en place : https://github.com/PrestaShop/PrestaShop/commit/c7ebf5ba5daaf54e7c1579c39f9d1d929f0259aa#diff-643b105ace43ea459d923f319583a84c

### Tout d'abord, `getContent()`.

**Avant**

```
public function getContent()
{
	$output = '<h2>'.$this->displayName.'</h2>';
	if (Tools::isSubmit('submitBlockCart'))
	{
		$ajax = Tools::getValue('cart_ajax');
		if ($ajax != 0 && $ajax != 1)
			$output .= '<div class="alert error">'.$this->l('Ajax : Invalid choice.').'</div>';
		else
			Configuration::updateValue('PS_BLOCK_CART_AJAX', (int)($ajax));
		$output .= '<div class="conf confirm">'.$this->l('Settings updated').'</div>';
	}
	return $output.$this->displayForm();
}
```

**Après**

```
public function getContent()
{
	$output = '';
	if (Tools::isSubmit('submitBlockCart'))
	{
		$ajax = Tools::getValue('cart_ajax');
		if ($ajax != 0 && $ajax != 1)
			$output .= $this->displayError($this->l('Ajax : Invalid choice.'));
		else
			Configuration::updateValue('PS_BLOCK_CART_AJAX', (int)($ajax));
		$output .= $this->displayConfirmation($this->l('Settings updated'));
	}
	return $output.$this->renderForm();
}
```

Les modifications sont minimes, la plupart important étant que nous n'utilisons plus directement de code HTML pour générer le contenu, mais utilisons plutôt les méthodes `displayError()` et `displayConfirmation()` pour respectivement afficher les messages d'erreur et ceux de réussite. Il n'est plus nécessaire d'inclure `displayName()` dans le code généré car celui-ci est pris en compte par les méthodes de PrestaShop.

Comme vous pouvez le voir, `getContent()` n'appelle plus `displayForm()`, mais à la place `renderForm()`. Nous avons simplement réécrit le code de `displayForm()` afin qu'il utilise les méthodes Helper, mais changer également le nom de la méthode participer à l'action de passer à la nouvelle méthode de construire des formulaires, mise en place par PrestaShop 1.5.

Voyons déjà ce à quoi ressemblait `displayForm()` :

```
public function displayForm()
{
	return '
	<form action="'.Tools::safeOutput($_SERVER['REQUEST_URI']).'" method="post">
		<fieldset>
			<legend><img src="'.$this->_path.'logo.gif" alt="" title="" />'.$this->l('Settings').'<
/legend>

			<label>'.$this->l('Ajax cart').'</label>
			<div class="margin-form">
				<input type="radio" name="cart_ajax" id="ajax_on" value="1" '
		.(Tools::getValue('cart_ajax', Configuration::get('PS_BLOCK_CART_AJAX')) ? 'checked="checked"
' : '').'/>
				<label class="t" for="ajax_on"> <img src="../img/admin/enabled.gif" alt="'
		.$this->l('Enabled').'" title="'.$this->l('Enabled').'" /></label>
				<input type="radio" name="cart_ajax" id="ajax_off" value="0" '
		.(!Tools::getValue('cart_ajax', Configuration::get('PS_BLOCK_CART_AJAX')) ? 'checked="
checked" ' : '').'/>
				<label class="t" for="ajax_off"> <img src="../img/admin/disabled.gif" alt="'
		.$this->l('Disabled').'" title="'.$this->l('Disabled').'" /></label>
				<p class="clear">'.$this->l('Activate AJAX mode for cart (compatible with the
default theme)').'</p>
			</div>

			<center><input type="submit" name="submitBlockCart" value="'.$this->l('Save').'" class="
button" /></center>
		</fieldset>
	</form>';
}
```

Et maintenant, voici la méthode `renderForm()`, qui fait plus ou moins la même chose que `displayForm()`, mais de manière plus propre, plus portable, et plus responsive. C'est de cette manière que vous devriez concevoir vos formulaires désormais.

⚠ Pour présentation complète de HelperForm, lisez la documentation : http://doc.prestashop.com/display/PS15/HelperForm

```
public function renderForm()
{
        $fields_form = array(
                'form' => array(
                        'legend' => array(
                                'title' => $this->l('Settings'),
                                'icon' => 'icon-cogs'
                        ),
                        'input' => array(
                                array(
                                        'type' => 'switch',
                                        'label' => $this->l('Ajax cart'),
                                        'name' => 'PS_BLOCK_CART_AJAX',
                                        'is_bool' => true,
                                        'desc' => $this->l('Activate AJAX mode for cart (compatible with the
default theme)'),
                                        'values' => array(
                                                array(
                                                        'id' => 'active_on',
                                                        'value' => 1,
                                                        'label' => $this->l('Enabled')
                                                ),
                                                array(
                                                        'id' => 'active_off',
                                                        'value' => 0,
                                                        'label' => $this->l('Disabled')
                                                )
                                        ),
                                )
                        ),
                'submit' => array(
                        'title' => $this->l('Save'),
                        'class' => 'btn btn-default pull-right')
                ),
        );

        $helper = new HelperForm();
        $helper->show_toolbar = false;
        $helper->table = $this->table;
        $lang = new Language((int)Configuration::get('PS_LANG_DEFAULT'));
        $helper->default_form_language = $lang->id;
        $helper->allow_employee_form_lang =
        Configuration::get('PS_BO_ALLOW_EMPLOYEE_FORM_LANG') ? Configuration::get
('PS_BO_ALLOW_EMPLOYEE_FORM_LANG') : 0;
        $this->fields_form = array();

        $helper->identifier = $this->identifier;
        $helper->submit_action = 'submitBlockCart';
        $helper->currentIndex = $this->context->link->getAdminLink('AdminModules', false)
       .'&configure='.$this->name.'&tab_module='.$this->tab.'&module_name='.$this->name;
        $helper->token = Tools::getAdminTokenLite('AdminModules');
        $helper->tpl_vars = array(
                'fields_value' => $this->getConfigFieldsValues(),
                'languages' => $this->context->controller->getLanguages(),
                'id_language' => $this->context->language->id
        );

        return $helper->generateForm(array($fields_form));
}
```

## Détails importants

### La variable 'bootstrap'

Une dernière chose à laquelle faire attention : si votre module utilise un contrôleur bootstrappé, vous devez ajouter la variable `bootstrap` à la méthode constructeur du module.

En effet, les helpers font certes le plus gros du travail, mais tant que vous n'indiquerez pas que vous utilisez Bootstrap, votre contrôleur sera encadré par des classes CSS "classiques", là où une seule ligne ajoutée lui fera utiliser les classes CSS "bootrstrappées" :

```
public function __construct()
{
        $this->bootstrap = true;
        $this->display = 'view';
        $this->meta_title = $this->l('Your Merchant Expertise');
        parent::__construct();
}
```

Cette ligne DOIT être placée dans la méthode `__construct()` de votre module – si vous utilisez des contrôleurs bootstrappés.

Si vous n'utilisez pas de contrôleurs bootstrappés, alors PrestaShop l'entourera d'une classe spécifique, qui fera de son mieux pour gérer le contrôleur de la meilleure manière possible, et ainsi assurer une certaine rétrocompatibilité.

**Taille de champs textuels**

Si vous souhaitez choisir la largeur de vos champs textuels, vous n'avez qu'à ajouter une classe à la balise input, directement le tableau décrit dans votre HelperForm.

Par exemple :

```
array (
        'type' => 'text',
        'label' => $this->l('Field name'),
        'name' => 'field_name',
        'class' => 'fixed-width-xs',       // Add this line.
        'required' => true
),
```

Dans cet exemple, nous utilisons "xs" comme largeur du champ.

Il a plusieurs tailles que vous pouvez utiliser :

- xs : très petite (*extra small*).
- sm : petite (*small*).
- md : moyenne (*medium*).
- lg : grande (*large*).
- xl : très grande *(extra large*).
- xxl : très très grande (*extra extra large*).