Making your module work with Bootstrap

Making your module work with Bootstrap

Version 1.6 of PrestaShop brings a whole new design to the default theme and the software itself. These designs are technically based on the Bootstrap 3 CSS framework (http://getbootstrap.com/), which enables designers and developers to rely on its tools and templates in order to create great and responsive designs.

As a module developer, you should strive to update your module to use Bootstrap, so that they integrate will into the new design.

It's all in the helpers

In itself, it is not complicated: most of the work is handled by PrestaShop's Helper methods, which have been upgrade to use Bootstrap the way it should be. Therefore, the hardest part for you is to move your module's interface code from the old way, where you defined your forms directly in HTML, into the new way that was introduced with PrestaShop 1.5, and which makes the Helpers methods do the heavy lifting.

Before / after

Here is an example of how your module can make use of the Bootstrap framework, through PrestaShop's Helper methods. This example is taken from the blockcart module, and concentrates on two methods: `getContent()` (which PrestaShop calls in order to display the module's configuration page), and `displayForm()` (which is replaced by `renderForm()`).

You can see the difference right on Github:

- Old blockcart module: https://github.com/PrestaShop/PrestaShop/blob/62ff976d69f4f5efd3413227f20bed429705e7b7/modules/blockcart/blockcart.php
- "Bootstrapped" blockcart module: https://github.com/PrestaShop/PrestaShop/blob/8144935d764d39d9ed809a1d16c8f452dd9f5591/modules/blockcart/blockcart.php
- The commit where all the work happens: https://github.com/PrestaShop/PrestaShop/commit/c7ebf5ba5daaf54e7c1579c39f9d1d929f0259aa#diff-643b105ace43ea459d923f319583a84c

First, `getContent()`.

Before

```
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();
}
```

After

```
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();
}
```

The changes are minimal, the most important one being that we do not use direct HTML code anymore to output content, but rather use PrestaShop's `displayError()` for error messages and `displayConfirmation()` for success messages. It is no longer necessary to include `displayName()` in the output since this it taken into account by PrestaShop methods.

As you can see, `getContent()` no longer calls on `displayForm()`, but rather `renderForm()`. We could simply rewrite `displayForm()`'s code to use the HelperForm methods, but changing the name of the method too helps to make the step to the new way of building forms since PrestaShop 1.5.

Let's first get a reminder of what `displayForm()` looked like.

```
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>';
}
```

Now, here is the `renderForm()` method, which makes pretty much the same thing as `displayFrom(`, but in a cleaner, more portable and now responsive way. This is how you should build forms from now on.

⚠ For a complete presentation of HelperForm, see this documentation page: 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));
}
```

Important details

The 'bootstrap' variable

One last thing to watch out for: if your module uses a bootstrapped controller, you must add the `bootstrap` variable to the module's constructor method.

Indeed, helpers do most of the hard work, but as long as you do not indicate that you are using Bootstrap, your controller will be surrounded by "classic" CSS classes, whereas a single line makes PrestaShop use the "bootstrapped" CSS classes:

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

This MUST be placed in your module's `__construct()` method to work – if you use bootstrapped controllers.

If you are not use a bootstrapped controller, then PrestaShop will wrap it with a specific class, which will do its best to handle the controller as effectively as possible, thus ensuring a certain level of retrocompatibility.

Text field width

If you want to choose the width of your text fields, just add a class on the input, directly in the array described in your HelperForm.

For instance:

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

In this example, "xs" is used to choose the width of the field.

There are several available sizes you can use:

- xs: extra small.
- sm: small.
- md: medium.
- lg: large.
- xl: extra large.
- xxl: extra extra large.