Creating a Dashboard Module -

Table of contents

Creating a Dashboard Module

Version 1.6 of PrestaShop brings a brand new Dashboard which features blocks of content that can be reorganized and, more importantly to you, added to.

Indeed, you can create your own dashboard modules, which you can make available for all to download or buy on PrestaShop Addons.

Differences with a regular PrestaShop module

A PrestaShop dashboard module is essentially the same thing as a regular PrestaShop module, with a few specifics. It is located in the `/modules` folder, and it can make use of PrestaShop's controllers, just like any module does.

The specifics

The name

Your dashboard module must have a unique name. As a convention, dashboard modules should all use the "dash" prefix: "dashproduct", "dashactivity", "dashgoals" are the names of some of the default dashboard modules

The constructor method

As for any PrestaShop module, the main PHP file must contain a `__construct()` method, which declares the usual variables: `name`, `displayName`, `version`, etc.

A few new variables are necessary for a proper dashboard module:

- `$this->push_filename`: the directory where your push data should be stored.
- `$this->allow_push`: a boolean indicating whether your module should have data be pushed to it. or not.
- `$this->push_time_limit`: the numbers of seconds between two data pushes.

Here is a sample `__construct()` method, taken from the Dashactivity module:

```
public function __construct()
{
        $this->name = 'dashactivity';
        $this->displayName = 'Dashboard Activity';
        $this->tab = '';
        $this->version = '0.1';
        $this->author = 'PrestaShop';
        $this->push_filename = _PS_CACHE_DIR_.'push/activity';
        $this->allow_push = true;
        $this->push_time_limit = 180;

        parent::__construct();
}
```

Note that the `tab` variable is empty. It is not needed as of today, but might be in future versions of PrestaShop.

The install() method

As for any PrestaShop module, the main PHP file must contain a `install()` method, which declares the usual hooks, such as `displayBackOfficeHeader`.

PrestaShop 1.6 implements several new hooks dedicated to dashboard modules:

- `dashboardData`: how you want your data to be handled.
- `dashboardZoneOne`: enables you to display your content in the left column of the dashboard zone.
- `dashboardZoneTwo`: enables you to display your content in the central column of the dashboard zone.

Here is a sample `install()` method, taken from the Dashproducts module:

```
public function install()
{
        if (!parent::install()
                || !$this->registerHook('dashboardZoneTwo')
                || !$this->registerHook('dashboardData'))
                return false;
        return true;
}
```

The zones

The dashboard has two available zones for your modules:

- Zone One: The left column of the dashboard.
- Zone Two: The central column of the dashboard.

Depending of the hooks your module is registered with, you might be able to display your content either on one of the two columns, or in both if need be.


The right column of the dashboard is not available for module.

The user cannot move a module from one column to the other, so you shouldn't hook your module to both zones.

Each hook must call on a template file in order to display your content.

Here is a sample `hookDashboardZoneTwo()` method, taken from the Dashproducts module:

```
public function hookDashboardZoneTwo($params)
{
        $this->context->smarty->assign(array(
                'date_from' => Tools::displayDate($params['date_from']),
                'date_to' => Tools::displayDate($params['date_to'])));
        return $this->display(__FILE__, 'dashboard_zone_two.tpl');
}
```

The templates

You should name your template after the zone it should be displayed in, as a handy reminder.

The template file should be stored in a module-specific folder: `/modules/dashmyodule/views/templates/hook/dashboard_zone_two.tpl`

The template itself is a classic PrestaShop template, with Smarty tags in HTML tags.
As you saw in the hook method, you can create new Smarty tags when your module is hooked to a zone. For instance, this code (inspired from the Dashactivity module):

```
public function hookDashboardZoneOne($params)
{
        $this->context->smarty->assign(array_merge(array(
                'dashactivity_config_form' => $this->renderConfigForm(),
                'date_subtitle' => $this->l('(from %s to %s)'),
                'date_format' => $this->context->language->date_format_lite
        ), $this->getConfigFieldsValues()));
        return $this->display(__FILE__, 'dashboard_zone_one.tpl');
}
```

This code declares three new tags:

- dashactivity_config_form: displays the content of the module's `renderConfigForm()` method.
- date_subtitle: displays a localized string ("from %s to %s").
- date_format: displays the date in "light" format (day.month.year).

From there on, you can call these tags directly in the template. For instance, `{$dashactivity_config_form}` displays the form in `renderConfigForm()`.

The data flow

The point of having a dashboard module is to display useful content to your user. When the Dashboard first loads, and every time you change a time setting (at the top of the Dashboard), PrestaShop triggers a call to the `dashboardData` hook in order to load data through Ajax requests.

In order to hook code the `dashboardData` hook, you must create a `hookDashboardData()` method, which must contain the SQL requests necessary for your data. This method must then return an array of the standard data types, each containing an array of the values in that data type.

For instance, here is the `hookDashboardData()` method from Dashproduct:

```
public function hookDashboardData($params)
{
    $table_recent_orders = $this->getTableRecentOrders();
    $table_best_sellers = $this->getTableBestSellers($params['date_from'], $params['date_to']);
    $table_most_viewed = $this->getTableMostViewed($params['date_from'], $params['date_to']);
    $table_top_10_most_search = $this->getTableTop10MostSearch($params['date_from'], $params['date_to']);
    return array(
        'data_table' => array(
            'table_recent_orders' => $table_recent_orders,
            'table_best_sellers' => $table_best_sellers,
            'table_most_viewed' => $table_most_viewed,
            'table_top_10_most_search' => $table_top_10_most_search,
        )
    );
}
```

Building hookDashboardData()

Your `hookDashboardData()` method can be as simple or complex as is necessary for your dashboard module. See for instance the way the Dashactivity module implements it: https://github.com/PrestaShop/PrestaShop/blob/c7fe91a0a9e02ca21183cc1c09e3e565d4de7265/modules/dashactivity/dashactivity.php#L95 The important thing is that it should return the various values as an array of arrays, with the data types as the root arrays.

See for instance the bottom of Dashactity's `hookDashboardData()` method:

```
return array(
    'data_value' => array(
        'pending_orders' => $pending_orders,
        'return_exchanges' => $return_exchanges,
        // etc.
    ),
    'data_trends' => array(
        'orders_trends' => array('way' => 'down', 'value' => 0.42),
    ),
    'data_list_small' => array(
        'dash_traffic_source' => $this->getReferer($params['date_from'], $params['date_to']),
    ),
    'data_chart' => array(
        'dash_trends_chart1' => $this->getChartTrafficSource($params['date_from'], $params['date_to']),
    ),
);
```

PrestaShop retrieves this data using an Ajax request, in JSON format:

```
{"dashactivity":{
  "data_value":{"pending_orders":"0","return_exchanges":"0"},
  "data_trends":{"orders_trends":{"way":"down","value":0.42}},
  "data_list_small":{"dash_traffic_source":{"Direct link":0}},
  "data_chart":{"dash_trends_chart1":{"chart_type":"pie_chart_trends","data":[{"key":"Direct link","y":0}]}}}
}
```

Note that you must set the values within an array corresponding to the data type:

- All Value arrays must be in the `data_value` array
- All Trends arrays must be in the `data_trends` array
- etc.

The data types

There are 4 data types that your module can use:

- Value: a single value, which can have any data type (string, number, boolean, etc.)
- Trends: a specific type, available as an array two values:
    - 'way': either 'up' or 'down', depending on the trend
    - 'value': the difference between the past and current value.
- List Small: an array of values.
- Chart: an array of two values:
    - 'chart_type': the type of chart to be used (ie.: 'pie_chart_trends').
    - 'data': an array of arrays:
        - key: the data key.
        - y: the data value.

Naming and getting the values displayed

The key for each value is very important, as this is how PrestaShop will update the display of your module: using JavaScript, the Dashboard will find the HTML tag which correspond to a value, and update its content.

For instance, this code in the Dashactivity's `dashboard_zone_one.tpl` file:

```
<span class="data_value size_l">
    <span id="pending_orders"></span>
</span>
```

...is tied to the JSON's `pending_orders` value through PrestaShop's JavaScript code – which you have yourself set in the PHP code for your `hookDashboardData()` method.

You must therefore pay attention to the way you name your HTML elements and your data key, as there is a clear correlation between the two within PrestaShop.

You own data types

In addition to the default ones, **you can create your own data type!** For instance, you could feel the need for a Slider data type, or a clear boolean. You return code would therefore be:

```
return array(
    'data_boolean' => array(
        'is_enabled' => getModuleState(),
        'must_auto_reload' => $auto_reload,
        // etc.
    ),
);
```

Which would give you the following JSON data response:

```
{"dashmymodule":{
  "data_boolean":{"is_enabled":true,"must_auto_reload":false},
}
```

The default data types have their data loaded and placed by specific JavaScript functions. In order for the data of your custom data types to still be loaded and correctly placed, you must add your own JavaScript function, which is called automatically when `hookDashboardData()` send its JSON back.

That JavaScript function should be in your own `.js` file, which you should add to the theme's header using the `$this->context->controller->addjs('js/mymethods.js', 'all');` method.

It should take this simple form:

```
function data_boolean(widget_name, data)
{
        // Here, the code that takes the data array sent by hookDashboardData, and handles their display in the
right location.
}
```

To get an example of such JavaScript methods, check the ones for `data_value`, `data_trends` and the other default data type in the file https://github.com/PrestaShop/PrestaShop/blob/e71094283395b092ccd0b0a0bb0a0fdfe25cbabc/js/admin-dashboard.js#L112.

The configuration form

Dashboard modules can have their own configuration form, which the user can access at the click of a button.

To declare the configuration form, you simply have to:

1. Declare the PHP function that will render this form.
2. Assign that function to a Smarty tag.
3. Call that Smarty tag in the template.

Here is how the Dashactivity module does it.

1 - Declaration:

```
public function renderConfigForm()
{
                $fields_form = array(
                                'form' => array(
                                                'id_form' => 'step_carrier_general',
                                                'input' => array(),
                                                'submit' => array(
                                                                'title' => $this->l(' Save '),
                                                                'class' => 'btn btn-default submit_dash_config',
                                                                'reset' => array(
                                                                                'title' => $this->l('Cancel'),
                                                                                'class' => 'btn btn-default
cancel_dash_config',
                                                                )
                                                )
                                ),
                );

                $sub_widget = array(
                                array('label' => $this->l('Show Pending'), 'config_name' =>
'DASHACTIVITY_SHOW_PENDING'),
                                array('label' => $this->l('Show Notifications'), 'config_name' =>
'DASHACTIVITY_SHOW_NOTIFICATION'),
                                array('label' => $this->l('Show Clients'), 'config_name' =>
'DASHACTIVITY_SHOW_CUSTOMERS'),
                                array('label' => $this->l('Show Newsletters'), 'config_name' =>
'DASHACTIVITY_SHOW_NEWSLETTER'),
                                array('label' => $this->l('Show Traffic'), 'config_name' =>
'DASHACTIVITY_SHOW_TRAFFIC'),
                );

// etc.
}
```

For the full `renderConfigForm()` code, see https://github.com/PrestaShop/PrestaShop/blob/c7fe91a0a9e02ca21183cc1c09e3e565d4de7265/modules/dashactivity/dashactivity.php#L297.

2 - Assignation to a Smarty tag:

```
public function hookDashboardZoneOne($params)
{
$this->context->smarty->assign(array_merge(array(
                'dashactivity_config_form' => $this->renderConfigForm(),
        ), $this->getConfigFieldsValues()));
        return $this->display(__FILE__, 'dashboard_zone_one.tpl');
}
```

3 - Use within the template:

```
<section id="dashactivity_config" class="dash_config hide">
        <header><i class="icon-wrench"></i> {l s='Configuration' mod='dashactivity'}</header>
        {$dashactivity_config_form}
</section>
```

Note that the template code should be exactly as presented here, so as to make sure that it fits well within the dashboard's design.