Understanding and using hooks

Understanding and using hooks

This article was written by Julien Breux, and published on May 5th, 2011 on the PrestaShop blog.

What is a "hook"?

As you've probably noticed, PrestaShop is a software that allows you to create modules designed to interact directly with the heart of the solution’s display or events.

The "hooks" allow you to retrieve these events or change the display.

So there are actually two distinct types of "hooks".

- Action "hooks" (letting you, for example, send out a mail when a client creates a user account)
- View "hooks" (enabling you, for example, to display a module in a column)

Sometimes the "hook" view can also serve as an action "hook." You do not need to do anything to display them. For example, you can make a recurring task on the homepage with the home hook.

If we take the example of the basic PrestaShop theme, on the home page, the solution uses "fixation points" as follows:

Hook name
header
top
leftColumn
home
rightColumn
footer

As you can see, all the "hooks" used are "view hooks".

This means that each of your modules can independently hang on to them and display information.

How do I use them?

First of all, to correctly use the "hook," you must go into your module's class and create a non-static, public method starting with the keyword "hook" and the name of "hook" used.
Then, only one single argument is passed: the array of different context information sent to the "hook".

```
public function hookNameOfHook($params)
{
}
```

Next, it is important when installing your module to stick it to various "hooks" desired. To do this, use the "registerHook" method, which also only accepts one parameter as well—the name of the hook.

```
public function install()
{
    return parent::install() && $this->registerHook('NameOfHook');
}
```

There is no need to use the module's "uninstall" method to remove the "hook".

Finally, it is important to understand how to call these "hooks" in order to create new ones later.

There are two calls in PrestaShop for "hooks." The second call is based on the first.

The first call is the direct method. It takes two arguments: the name of the "hook" and an array of different context information.

```
$params = array(
    'param_1' => 'value_1',
    'param_2' => 'value_2'
);
Module::hookExec('NameOfHook', $params);
```

The second call is a "shortcut" to the first in order to have a "cleaner" display when making the call. The set of "shortcuts" are available in the class "Hook."

```
class HookCore extends ObjectModel
{
    // ...
    static public function updateProduct($product)
    {
        $params = array('product' => $product);
        return Module::hookExec('updateProduct', $params);
    }
    // ...
}
```

Calling the "hook" called "updateProduct" will therefore be as follows in the heart of PrestaShop.

```
Hook::updateProduct(new Product(/* ... */));
```

We have called the class "HookCore" using "Hook" as the class name. This is due to the override that we will go over next time!
How to add new one?

How to add new one?

If so far you've followed how to use "hooks" and how to use them in PrestaShop, then you'll surely you realize that you're missing some "hooks" like "MakeCoffe", "MakeFood" or "RespondToClientsForMe". Don't panic!

To create your own little personal "hook", you just simply save a row in the database table "ps_hook" with the name of your "hook". (See 0 and 1 if it is compatible with LiveEdit or not)

```
INSERT INTO `ps_hook` (`name`, `title`, `description`)
              VALUES ('nameOfHook', 'Name Of Hook', 'It is a custom hook !');
```

And finally, just use it as we have done in this article !