Using the Context Object

Table of contents

Using the Context Object

What is the Context object?

The Context is a technical feature introduced with version 1.5 of PrestaShop. Its two goals are:

- preventing developers from using global variables.
- enabling them to change the context of some methods.

The Context is a registry for PHP variables that were previously accessed as globals. It aims to standardize the way these variables are accessed, and to make the code more robust by getting rid of global vars.
It is a light implementation of the Registry design pattern: it is a class that stores the main PrestaShop information, such as the current cookie, the customer, the employee, the cart, Smarty, etc.

Before version 1.5, you had to rely on the cookie in order to access this data:

PrestaShop 1.4 code

```
$cookie->id_lang;
```

Now that the Context is available, the same data can be accessed more cleanly:

PrestaShop 1.5 code

```
$this->context->language->id;
```

What is stored by the Context?

These objects are always accessible through the context:

- **Language**. Set with the customer or employee language.
- **Country**. Default country.
- **Currency**. Set with the customer currency or the shop's default currency.
- **Shop**. Current shop.
- **Cookie**. Cookie instance.
- **Link**. Link instance.
- **Smarty**. Smarty instance.

These objects are only accessible for the customer Context:

- **Customer**. Existing customer retrieved from the cookie or default customer.
- **Cart**. Current cart.
- **Controller**. Current controller instance.

These objects are only accessible for the administrator Context:

- **Employee**. Current employee.

How to access the Context?

From inside a `Controller` subclass, an `AdminTab` subclass or a `Module` subclass, the Context should be called with this shortcut: `$this->context`.

From anywhere else, you can get the Context instance by calling `Context::getContext()`.

How is the Context initialized?

The context is initialized with data coming from the cookie or from the database. For example, to create the `Language` object, the context looks for an `id_lang` value in the cookie. If it doesn't find one, it will retrieve the default language `id` from the database.

How to use the Context?

Whenever you would have accessed the cookie to retrieve a variable, or used a global statement, you will probably want to access the Context instead.

A few common replacements:

- Static shortcuts in `FrontController`subclasses are deprecated:

PrestaShop 1.4 code

```
$id_cart = self::cart->id;
```

...is to be replaced with...

PrestaShop 1.5 code

```
$id_cart = $this->context->cart->id;
```

- Do not recreate an object that already exists:

PrestaShop 1.4 code

```
$language = new Language($cookie->id_lang);

$iso_code = $language->iso_code;
```

...is to be replaced with...

PrestaShop 1.5 code

```
$iso_code = $this->context->language->iso_code;
```

More examples of Context use

Old way	New way
$cookie->id_lang;	$this->context->language->id;
if ($cookie->isLogged())	if ($this->context->customer->isLogged())
if ($cookie->isLoggedBack())	if ($this->context->employee->isLoggedBack();
$cart->getProducts();	$this->context->cart->getProducts();
$language = new Language($cookie->id_lang); $language->iso_code;	$this->context->language->iso_code;
new Currency($cookie->id_lang);	$this->context->currency;
$defaultCountry->id_zone;	$this->context->country->id_zone;
new Link();	$this->context->link;
$smarty->assign(...);	$this->context->smarty->assign(...);

Using the Context in a PrestaShop 1.4 module

Modules written for PrestaShop 1.5, and which therefore rely on the Context object, can be made to gracefully degrade its Context use in order to work in PrestaShop 1.4.

Add this method to the module's main class:

```
// Retrocompatibility 1.4/1.5
private function initContext()
{
  if (class_exists('Context'))
    $this->context = Context::getContext();
  else
  {
    global $smarty, $cookie;
    $this->context = new StdClass();
    $this->context->smarty = $smarty;
    $this->context->cookie = $cookie;
  }
}
```

Then, add this line in the module's constructor method:

```
// Retrocompatibility
$this->initContext();
```

Additionally, when rewriting you v1.5 module in work to work in v1.4, you should use the older hook names, such as `displayProductTab` or `displayProductTabContent`. These two example hooks are respectively registered using the `productTab` and `productTabContent` calls, which are compatible for both v1.4 and v1.5, but will not work in the next major version.