Setting Up Your Local Development Environment -

Table of contents

Setting Up Your Local Development Environment

Now that you intend to develop for PrestaShop, you are better off keeping all your development work on your machine. The main advantage is that it makes it possible for you to entirely bypass the process of uploading your file on your online server in order to test it. Another advantage is that a local test environment enables you to test code without the risk of breaking your production store. Have a local environment is the essential first step in the path of web development.

Installing PrestaShop locally

Prerequisites

PrestaShop needs the following server configuration in order to run:

- System: Unix, Linux or Windows.
- Web serve: Apache Web Server 1.3 or any later version.
- PHP: 5.2 or later.
- MySQL: 5.0 or later.
- At least 32 Mb of RAM. 64 Mb is more comfy, the more the better...

⚠ PrestaShop can also work with Microsoft's IIS Web server 6.0 or later, and nginx 1.0 or later.

Installing a local environment

Installing any web-application locally requires that you first install the adequate environment, namely the Apache web server, the PHP language interpreter, the MySQL database server, and ideally the phpMyAdmin tool. This is called an AMP package: Apache+MySQL+PHP and the operating system, giving WAMP (Windows+Apache+MySQL+PHP), MAMP (Mac OS X+...) and LAMP (Linux+...). Since all of the items packaged are open-source, these installers are most of the time free.

Here is a selection of free AMP installer:

- XAMPP: http://www.apachefriends.org/en/xampp.html (Windows, Mac OS X, Linux, Solaris)
- WampServer: http://www.wampserver.com/en/ (Windows)
- EasyPHP: http://www.easyphp.org/ (Windows)

- MAMP: http://www.mamp.info/ (Mac OS X)

Configuring PHP

PrestaShop needs a few additions to PHP and MySQL in order to fully work. Make sure that your PHP configuration has the following settings and tools:

- GD library.
- Dom extension.
- `allow_url_fopen` enabled.

Here is a section of the `php.ini` file (the configuration file for PHP):

```
extension = php_mysql.dll
extension = php_gd2.dll
allow_url_fopen = On

# also recommended
register_globals = Off
magic_quotes_gpc = Off
allow_url_include = Off
```

⚠ The GD library (http://www.boutell.com/gd/) enables PrestaShop to rework images that you upload, especially resizing them.

The Dom extension enables to parse XML documents. PrestaShop uses it for various functionalities, like the Store Locator. It is also used by some modules, as well as the `pear_xml_parse` library.

The `allow_url_fopen` directive enables modules to access remote files, which is an essential part of the payment process, among others things. It is therefore imperative to have it set to **ON**.

Downloading and extracting the PrestaShop files

You can download the latest version of PrestaShop at http://www.prestashop.com/en/downloads.

You can download the (unstable) development version on Github: https://github.com/PrestaShop/PrestaShop/archive/development.zip

Extract the PrestaShop files, and put them in the root folder of the AMP installer you chose:

- XAMPP: `C:\xampp\htdocs` or `/Applications/xampp/htdocs`
- WampServer: `C:\wamp\www`
- EasyPHP: `C:\easyphp\www`
- MAMP: `/Applications/MAMP/htdocs/`

Creating a database for your local shop

Open the phpMyAdmin tool using your browser. Its location depends on the AMP pack you chose:

- `http://127.0.0.1/phpmyadmin` (XAMPP, WampServer, MAMP),
- `http://127.0.0.1/mysql` (EasyPHP)

In the "Databases" tab, indicate the database name you want and validate by clicking on the "Create a database" button.

Installing PrestaShop

Open the PrestaShop installer, which should be located at `http://127.0.0.1/prestashop/install`, and follow its instructions.

You can read the Getting Started guide for more details: http://doc.prestashop.com/display/PS16/Getting+Started.

Configuring PrestaShop

By default, PrestaShop is configured to provide a secure and stable environment to both the shop administrator and the customers.

As a developer, there are several changes that you could and should bring to the default installation in order to help you code better, spot bugs faster, and generally make a great PrestaShop product.

Disabling the cache and forcing Smarty compilation

When your development has an impact on the front office, whether you are building a theme or simply a module which displays information to the customer, you should force the template file compilation and disable the cache, so as to always see the result of your changes directly.

Go to the "Performances" page under the "Advanced parameters" menu to change the following Smarty settings:

- Template cache: switch it to "Force compilation".
- Cache: disable it.

> Forcing the compilation of Smarty will always slow down the loading time of the page. Make sure that your production store is set to only recompile templates if there are updated files, and that its cache is enabled.

Displaying error messages

PrestaShop's default settings prevent the customer to see any server error message or any debugging code.

You, on the other hand, need this information in order to correct any potential mistake in your code. To that end, open the `/config/defines.inc.php` file, and edit it to set `_PS_MODE_DEV_` to `true`:

```
/* Debug only */
define('_PS_MODE_DEV_', true);
```

Using the debug methods

PrestaShop has custom debug methods available for developers: `p($variable)` and `d($variable)`. They are used to display the content of a variable. It is really a wrapper around the well-known `print_r()` method (http://php.net/manual/en/function.print-r.php):

The p() method

```
echo '<xmp style="text-align: left;">';
print_r($variable);
echo '</xmp><br />';
return $variable;
```

This type of function is typical of many PHP development sessions, and PrestaShop enables you to not have to reinvent the wheel with this method.

`p()` is the main method, and `d()` works the same way, except that it calls the `die()` method instead of returning the variable:

The d() method

```
echo '<xmp style="text-align: left;">';
print_r($variable);
echo '</xmp><br />';
die('END');
```

These two methods enable you to check for the state of a given variable at a specific place within your code.

On top of that, PrestaShop defines the `ppp()` and `ddd()` methods, which are respectively the aliases of `p()` and `d()`. They work exactly the same, but are often easier to search and find in a huge block of code.

These debug methods are not activated by default. To activate them, you must enable the Debug mode, by setting `_PS_MODE_DEV_` to `true` (see above).

Enabling the multistore mode

PrestaShop 1.6 is able to host more than one store within a single installation of the software. Many shop administrators choose to enable this feature, and it can have a significant impact on the way PrestaShop works. You should therefore make sure that anything you code for PrestaShop works in both single and multistore mode.

Enabling the multistore mode is easy: go to the general preferences page, and put the "Enable Multistore" option to "Yes".

You can switch back and forth between single store and multistore mode – in single store mode, only the main store is used.

You can read more about the multistore mode in the PrestaShop 1.6 User Guide: http://doc.prestashop.com/display/PS16/Managing+Multiple+Shops.

About the configuration files

There are three main configuration files, all located in the `/config` folder:

- `config.inc.php`
- `defines.inc.php`
- `smarty.config.inc.php`

config.inc.php

It is the main configuration file for PrestaShop. You should not have to touch anything in there.

defines.inc.php

This file contains PrestaShop constant values.

It also contains the location of all the files and folders. If you need to change their location, do not forget to keep the original path nearby, for instance in a PHP comment, in case you need to revert back to it later on.

When in development/test mode, you must make sure that all the error messages are displayed:

- Set `define('_PS_MODE_DEV_', false);` to `true`.

On the contrary, when in production mode, you must hide error messages as much as possible!

- Make sure that `define('_PS_MODE_DEV_', false);` is set to `false`.

smarty.config.inc.php

This file contains all the Smarty-related settings.

The Smarty cache system should always be disabled, as it is not compatible with PrestaShop: keep `$smarty->caching = false;` as it is.

`$smarty->compile_check` should be left to `false` in development mode.

`$smarty->debugging` gives access to Smarty debug information when displaying a page. That setting is more easily modified in the "Performance" page of the advanced parameters menu : the "Debug console" option enables you to choose between never displaying Smarty's debug information, always displaying it, or only displaying it when you add `?SMARTY_DEBUG` to the URL of the page you want to test, which can be very useful.

> When in production mode, `$smarty->force_compile` must be set to `false`, as it will give a 30% boost to your page load time.
>
> On the other hand, when editing a `.tpl` file, you must delete the `/tools/smarty/compile` folder (except the `index.php` file) in order to see your changes applied.
>
> Note that this setting can be made directly from the back office, in the "Performance" page under the "Advanced parameters" menu.

Keeping things secure

Once your module is online, its files could be accessed by anyone from the Internet. Even if they cannot trigger anything but PHP errors, you might want to prevent this from happening.

You can achieve this by adding a `index.php` file at the root of any module folder you create. Here is a suggestion for what to put in the file.

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: ".gmdate("D, d M Y H:i:s")." GMT");
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache");
header("Location: ../");
exit;
```