Asset Management

PrestaShop 1.7 has significantly improved the way assets (CSS, JavaScript and image files) are managed.

We advise theme developers to compile most of their style and JavaScript code into a single concatenated/minified file (see the Webpack section below).

If you need to add special assets, for example an extra JavaScript library on the home page or the product page, there are a few ways to do so.

Your theme have to print assets correctly in the smarty template, and it’s explained in the template section.

Registering assets

In PrestaShop 1.7+, it’s easy to register custom assets on each pages. The major improvement is that you can easily manage them from your theme, without any modules.

We introduced new methods to register assets, and especially new cool options.

For instance, you can register your asset specifically in the head or bottom of your HTML code; you can load it with attributes like `async` or `defer`; and you can even inline it easily.

One favorite option is the *priority* one, which makes it very easy to ensure everything is loaded in the order you need.

Backward compatibility is kept for the `addJS()`, `addCSS()`, `addJqueryUI()` and `addJqueryPlugin()` methods. Incidentally, now is the best time to update your libraries and use the new method.

Here is a list of options, and what they do.

Options

PrestaShop’s *FrontController* class provides 2 new methods to easily register new assets: `registerStylesheet()` and `registerJavascript()`.

In order to have the most extensible signatures, these 2 methods take 3 arguments. The first one is the unique ID of the asset, the second one is the relative path, and the third one is an array of all other optional parameters, as described below.

ID

This unique identifier needed for each asset. This is useful to either override or unregister something already loaded by the Core or a native module.

Relative path

This is the path of your asset. In order to make your assets fully overridable and compatible with the parent/child feature, you need to provide the path from the theme's root directory, or PrestaShop's root directory if it's a module.

For example:

- 'assets/css/example.css' for something in your theme.
- 'modules/modulename/css/example.cs' for something in your module.

Extra parameters for stylesheet

Name	Values	Comment
media	all|braille|embossed|handheld|print|projection|screen|speech| tty|tv (default: all)	no comment.
priority	0-999 (default: 50)	0 is the highest priority
inline	true|false (default: false)	If true, your style will be printed inside the `<style>` tag in your HTML `<head>`. Use with caution.

Extra parameters for JavaScript

Name	Values	Comment
position	head|bottom (default: bottom)	JavaScript files should be loaded in the bottom as much as possible. Remember: core.js is loaded first thing in the bottom so jQuery won't be loaded in the <head> part.
priority	0-999 (default: 50)	0 is the highest priority
inline	true|false (default: false)	If true, your style will be printed inside `<script type="text/javascript">` tags inside your HTML. Use with caution.
attribute	async|defer|none (default: none)	Load JavaScript file with the corresponding attribute (Read more: Async vs Defer attributes)

Registered by the Core

Every page of every theme loads the following files:

- theme.css
- custom.css
- rtl.css (if a right-to-left language is detected)
- core.js
- theme.js
- custom.js

Filename	ID	Priority	Comment
theme.css	theme-main	0	Most (all?) of your theme's styles. Should be minified.
rtl.css	theme-rtl	900	Loaded only for Right-To-Left language
custom.css	theme-custom	1000	Empty file loaded at the very end to allow user to override some simple style.
core.js	corejs	0	Provided by PrestaShop. Contains Jquery2, dispatches PrestaShop events and holds PrestaShop logic.

theme.js	theme-main	50	Most of your theme’s JavaScript. Should embed librariesrequired on all pages, and be minified.
custom.js	theme-custom	1000	Empty file loaded at the very end, to allow user to override behavior or add simple script.

Registering in themes

By now you probably understood that this `theme.yml` file became the heart of PrestaShop themes.

To register assets, create a new `assets` key at the top level of your `theme.yml`, and register your files according to your needs. Page identifiers are based on the `php_self` property of each controller (example)

For example, if you want to add an external library on each page and a custom library on the Product page:

```
assets:
  css:
    product:
      - id: product-extra-style
        path: assets/css/product.css
        media: all
        priority: 100
  js:
    all:
      - id: this-cool-lib
        path: assets/js/external-lib.js
        priority: 30
        position: bottom
    product:
      - id: product-custom-lib
        path: assets/js/product.js
        priority: 200
        attribute: async
```

Registering in modules

When developing a PrestaShop module, you may want to add specific styles for your templates. The best way is to use the `registerStylesheet` and `registerJavascript` methods provided by the parent `FrontController` class.

⚠ If you’re developing a custom module that only works on your themes, don’t put any style or JavaScript code inside the module: put it in the theme’s files instead (`theme.js` and `theme.css`).

With a front controller module

If you develop a front controller, simply extend the `setMedia()` method. For instance:

```
public function setMedia()
{
    parent::setMedia();

    if ('product' === $this->php_self) {
        $this->registerStylesheet(
            'module-modulename-style',
            'modules/'.$this->module->name.'/css/modulename.css',
            [
              'media' => 'all',
              'priority' => 200,
            ]
        );

        $this->registerJavascript(
            'module-modulename-simple-lib',
            'modules/'.$this->module->name.'/js/lib/simple-lib.js',
            [
              'priority' => 200,
              'attribute' => 'async',
            ]
        );
    }
}
```

Without a front controller module

If you only have your module's class, register your code on the *actionFrontControllerSetMedia* hook, and add your asset on the go inside the hook:

```
public function hookActionFrontControllerSetMedia($params)
{
    // Only on product page
    if ('product' === $this->context->controller->php_self) {
        $this->context->controller->registerStylesheet(
            'module-modulename-style',
            'modules/'.$this->name.'/css/modulename.css',
            [
              'media' => 'all',
              'priority' => 200,
            ]
        );

        $this->context->controller->registerJavascript(
            'module-modulename-simple-lib',
            'modules/'.$this->name.'/js/lib/simple-lib.js',
            [
              'priority' => 200,
              'attribute' => 'async',
            ]
        );
    }

    // On every pages
    $this->context->controller->registerJavascript(
        'google-analytics',
        'modules/'.$this->name.'/ga.js',
        [
          'position' => 'head',
          'inline' => true,
          'priority' => 10,
        ]
    );
}
```

Unregistering

You can unregister assets! That’s the whole point of an `id`. For example if you want to improve your theme/module’s compatibility with a module, you can unregister its assets and handle them yourself.

Let’s say you want to be fully compatible with a popular navigation module. You could create a template override of course, but you could also remove the style that comes with it and bundle your specific style in your `theme.css` (since it’s loaded on every page).

To unregister an assets, you need to know its ID.

In themes

As of today, the only way to unregister an asset without any module is to place an empty file where the module override would be.

If the module registers a JavaScript file placed in `views/js/file.js`, you simply need to create an empty file in `modules/modulename/views/js/file.js`.

It works for both JavaScript and CSS assets.

In modules

Both `unregisterJavascript` and `unregisterStylesheet` methods take only one argument: the unique ID of the resource you want to remove.

```
// In a front controller
public function setMedia()
{
    parent::setMedia();

    $this->unregisterJavascript('the-identifier');
}

// In a module class
public function hookActionFrontControllerSetMedia($params)
{
  $this->context->controller->unregisterJavascript('the-identifier');
}
```

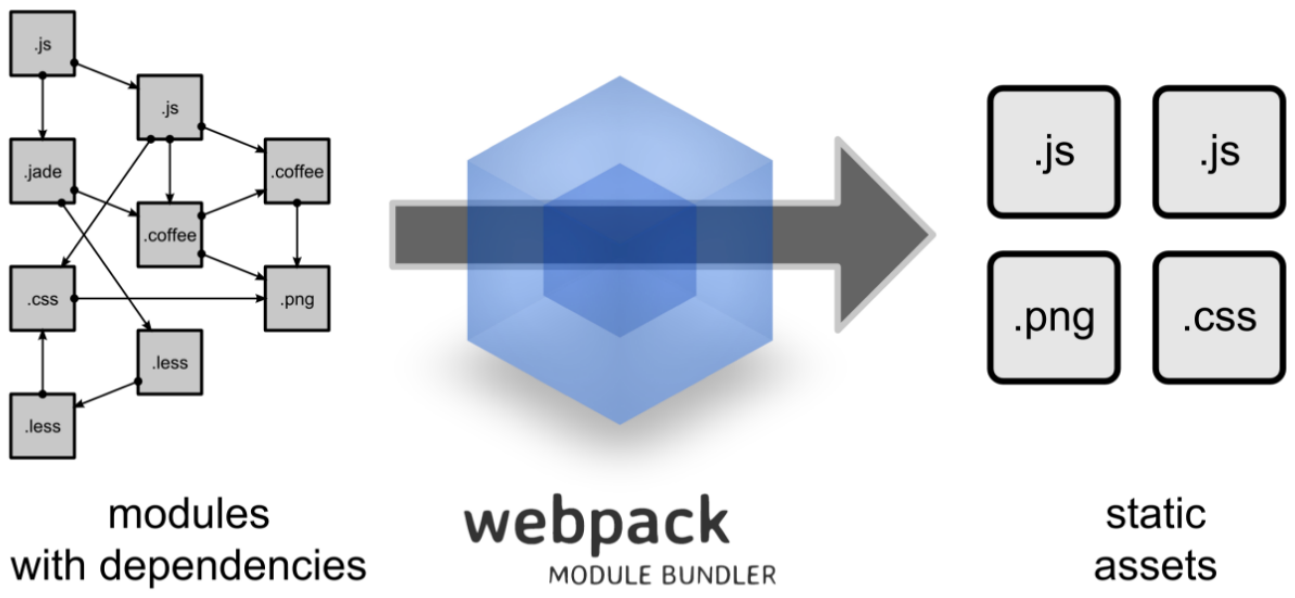
About Webpack

ⓘ Webpack is a module bundler. Webpack takes modules with dependencies and generates static assets representing those modules.

The main interest of using Webpack is that it will compile all your styles - which we advise you to write using Sass - into a single CSS file. This way, your theme will make only one HTTP request for this single file, and since your browser will cache it for later re-use, it will even donwload this file only once.

The same goes with your JavaScript code. Instead of loading jQuery along with its community plugins, your own custom plugins and any extra code you might need, Webpack compiles and minifies all this JavaScript code into a single file, which will be loaded once - and cached.

⚠ Webpack is not at all required by PrestaShop, you are free to use your favorite tool! The documentation explains Webpack since it’s the tool we chose for the Classic theme, and StarterTheme ships with a ready-to-use configuration file.



Installing webpack

If you want to compile your assets using Webpack (and we advise you to), follow these steps:

1. Download and install Node.js, which contains the npm tool.
2. In your command line tool, open the *_dev* folder.
3. Install npm: *npm install*.
4. You then have a choice:
    - To build your assets once, type *npm run build*.
    - To rebuild your assets every time you change a file in the _dev folder, type *npm run watch*.

Webpack configuration

The Webpack configuration file for StarterTheme is thus:

1. All CSS rules go to the *assets/css/theme.css* file.
2. All JavaScript code go to the *assets/js/theme.js* file.

It provides proper configuration for compile your Sass, Less, Stylus or CSS files into a single CSS file.

JavaScript code is written in ES6, and compiled to ES5 with Babel.

If you want to use Stylus or Less, simply edit the command line under the “scripts” section.

```
var webpack = require('webpack');
var ExtractTextPlugin = require("extract-text-webpack-plugin");

var plugins = [];

plugins.push(
  new ExtractTextPlugin('../css/theme.css')
```

```
);

module.exports = [{
  // JavaScript
  entry: [
    './js/theme.js'
  ],
  output: {
    path: '../assets/js',
    filename: 'theme.js'
  },
  module: {
    loaders: [{
      test: /\.js$/,
      exclude: /node_modules/,
      loaders: ['babel-loader']
    }]
  },
  externals: {
    prestashop: 'prestashop'
  },
  plugins: plugins,
  resolve: {
    extensions: ['', '.js']
  }
}, {
  // CSS
  entry: [
    './css/normalize.css',
    './css/example.less',
    './css/st/dev.styl',
    './css/theme.scss'
  ],
  output: {
    path: '../assets/js',
    filename: 'theme.js'
  },
  module: {
    loaders: [{
      test: /\.scss$/,
      loader: ExtractTextPlugin.extract(
        "style",
        "css-loader?sourceMap!postcss!sass-loader?sourceMap"
      )
    }, {
      test: /\.styl$/,
      loader: ExtractTextPlugin.extract(
        "style",
        "css-loader?sourceMap!postcss!stylus-loader?sourceMap"
      )
    }, {
      test: /\.less$/,
      loader: ExtractTextPlugin.extract(
        "style",
        "css-loader?sourceMap!postcss!less-loader?sourceMap"
      )
    }, {
      test: /\.css$/,
      loader: ExtractTextPlugin.extract(
        'style',
        'css-loader?sourceMap!postcss-loader'
      )
    }, {
      test: /.(png|woff(2)?|eot|ttf|svg)(\?[a-z0-9=\.]+)?$/,
      loader: 'file-loader?name=../css/[hash].[ext]'
    }]
  },
  plugins: plugins,
  resolve: {
    extensions: ['', '.scss', '.styl', '.less', '.css']
  }
}];
```