



PrestaShop 1.7

Développeur de modules Back-Office

| | |
|--|----|
| Développeur de module back-office | 13 |
| PrestaShop | 14 |
| PrestaShop | 14 |
| Les dernières versions | 14 |
| Numérotation de version | 14 |
| Prestashop 1.5 | 15 |
| Technologies avec la 1.5..... | 15 |
| Prestashop 1.6 | 16 |
| Technologies avec la 1.6..... | 16 |
| Prestashop 1.7 | 17 |
| Technologies avec la 1.7..... | 17 |
| Les nouveautés de la 1.7 | 17 |
| Nouveautés en intégration..... | 18 |
| Nouveautés en développement | 18 |
| Dernières suppressions..... | 18 |
| Nouvelles normes..... | 18 |
| Technologies utilisées par PrestaShop..... | 19 |
| Sass | 19 |
| Compass | 19 |
| Bourbon..... | 19 |
| Bootstrap 3..... | 20 |
| Bootstrap 4..... | 20 |
| Smarty..... | 20 |
| Twig..... | 21 |

| | |
|--|----|
| Symfony | 21 |
| Outils | 21 |
| Composer..... | 21 |
| Node.js | 22 |
| Webpack | 22 |
| Git | 22 |
| GitHub | 23 |
| Le validateur de modules | 23 |
| Le générateur de modules..... | 24 |
| Documentation et Support | 24 |
| Documentation | 24 |
| Development blog | 24 |
| Les forums de discussion | 25 |
| Slack | 25 |
| Installation et Configuration | 26 |
| Installation de PrestaShop | 26 |
| Les pré-requis | 26 |
| Version de PHP | 26 |
| Mettre en place un environnement local | 26 |
| Configurer PHP | 27 |
| Installation avec git..... | 27 |
| Installation par téléchargement..... | 27 |
| Contenu du fichier release..... | 27 |
| Autres techniques d'installation | 28 |
| Créer une base de données. | 28 |
| Installation à partir d'un navigateur. | 28 |
| Installation par ligne de commande. | 28 |
| Configuration | 29 |
| Compilation des templates Smarty | 29 |
| Activer l'affichage des erreurs..... | 29 |
| Utiliser Nginx | 29 |
| Système de cache..... | 29 |
| Environnement de travail | 30 |
| Editeur..... | 30 |
| Composer..... | 30 |
| Utiliser Composer..... | 30 |
| Node.js..... | 31 |
| Utiliser npm | 31 |
| Xdebug..... | 31 |

| | |
|--|----|
| Exercice | 32 |
| Exercice | 32 |
| Architecture | 33 |
| Environnements | 33 |
| Le front-office | 33 |
| Le back-office legacy | 33 |
| Le back-office modern | 33 |
| Architecture de PrestaShop | 34 |
| Architecture MVC | 34 |
| Modèles..... | 34 |
| Vues | 34 |
| Contrôleurs..... | 34 |
| Autoload | 35 |
| Stratégie de migration | 35 |
| Nouvelle architecture CQRS..... | 35 |
| Nouvelle architecture DDD | 35 |
| Liste des répertoires | 36 |
| Répertoire modules | 36 |
| Répertoire mails | 37 |
| Répertoire img..... | 37 |
| Répertoire pdf..... | 38 |
| Répertoire themes..... | 38 |
| Nouveautés de la 1.7..... | 38 |
| Méthodes | 39 |
| Dependency injection | 39 |
| Avec dependency injection | 39 |
| Framework | 40 |
| La notion de contexte..... | 40 |
| Les objets stockés dans le contexte | 40 |
| Utilisation du context | 40 |
| Les contrôleurs | 41 |
| Fonctionnement d'un contrôleur | 41 |
| La classe ModuleFrontController | 41 |
| La classe ModuleAdminController | 41 |
| La surcharge d'un contrôleur. | 42 |
| Les vues | 42 |
| Les templates Smarty..... | 42 |
| Les templates Smarty utilisés sur le front-office..... | 42 |
| Les templates Smarty utilisés sur le back-office (Legacy)..... | 42 |

| | |
|---|----|
| La surcharge d'une vue sur le front-office | 43 |
| La surcharge d'une vue sur le back-office..... | 43 |
| Les cookies..... | 43 |
| Utiliser les cookies dans PrestaShop..... | 43 |
| Informations stockées dans un Cookie Visiteur | 43 |
| Informations stockées dans un Cookie Employé | 44 |
| Utiliser les cookies - depuis un script indépendant de PrestaShop | 44 |
| Classes du framework..... | 45 |
| PrestaShopAutoload..... | 45 |
| Dispatcher | 45 |
| ImageManager | 45 |
| Link | 45 |
| Media | 46 |
| PhpEncryption..... | 46 |
| ProductAssembler | 46 |
| Tools..... | 46 |
| Translate | 47 |
| Uploader..... | 47 |
| Validate | 47 |
| Les Helpers..... | 48 |
| Les Helpers pour le back-office legacy | 48 |
| Les templates des Helpers | 48 |
| Utiliser les Helpers | 48 |
| Paramétrage des Helpers..... | 48 |
| Nouveautés de la 1.7. | 49 |
| Espace de noms (namespaces) | 49 |
| Espace PrestaShop\PrestaShop\Core..... | 49 |
| Espace PrestaShop\PrestaShop\Adapter | 49 |
| Accéder au nouveau code depuis l'ancien..... | 49 |
| Accéder à l'ancien code depuis le nouveau | 50 |
| Tunnel de commande | 50 |
| Contrôleurs Front | 50 |
| Formulaires front-office..... | 51 |
| Fonctionnement des formulaires sur front-office | 51 |
| Les formulaires existants sur le front-office..... | 51 |
| API pour les modules de paiement..... | 51 |
| Exercice | 52 |
| Solution | 53 |
| Symfony | 54 |

| | |
|--|----|
| Répertoire de Symfony | 54 |
| Composants Symfony | 54 |
| Service container | 54 |
| Principe | 54 |
| Disponibilité | 55 |
| Les services de Symfony | 55 |
| Les services de PrestaShop | 55 |
| La console Symfony | 55 |
| Principe | 55 |
| Les commandes PrestaShop | 55 |
| La commande prestashop:module | 56 |
| La commande cache:clear | 56 |
| Data Access | 57 |
| Infrastructure | 57 |
| Système de gestion de base de données (SGBD) | 57 |
| Configuration recommandée | 57 |
| Optimiser MySQL / MariaDB | 57 |
| Plusieurs bases de données | 57 |
| Exemple d'infrastructure | 58 |
| Schéma de la base de données | 58 |
| Diagramme | 58 |
| Encodage des caractères | 59 |
| Le préfixe des tables | 59 |
| Le moteur de stockage | 59 |
| Choisir un moteur de stockage | 59 |
| Création de la base de données | 59 |
| Mise à jour de la base de données | 59 |
| Conventions de nommage | 60 |
| Normes pour le nom des tables | 60 |
| Suffixe _lang | 60 |
| Suffixe _shop | 60 |
| Normes pour le nom des champs | 60 |
| id_lang et id_shop | 60 |
| DB | 60 |
| La classe DB | 60 |
| Gestion du cache | 61 |
| Fonction getInstance() | 61 |
| Fonction query() | 61 |
| Fonction insert() | 62 |

| | |
|--|----|
| Fonction update() | 62 |
| Fonction delete() | 62 |
| Fonction execute() | 62 |
| Fonction executeS() | 63 |
| Fonction getRow() | 63 |
| Fonction getvalue() | 63 |
| Fonction numRows() | 64 |
| Fonction escape() | 64 |
| Les fonctions d'interface avec PDO et MySQLi | 65 |
| Autres fonctions. | 65 |
| ObjectModel | 66 |
| La Classe ObjectModel | 66 |
| Classe de type Active Record | 66 |
| Utilisation | 66 |
| La définition du modèle | 66 |
| Les différents types de zone | 67 |
| Pour un objet en plusieurs langues | 67 |
| Pour un objet en fonction de la boutique et en plusieurs langues | 67 |
| DBQuery | 67 |
| La classe DBQuery | 67 |
| Les méthodes | 68 |
| Doctrine | 68 |
| Le mapping | 68 |
| Les entités | 69 |
| PrestaShop et Doctrine | 69 |
| Le service EntityManager | 69 |
| Les Repository | 70 |
| Créer un enregistrement | 70 |
| Modifier un enregistrement | 70 |
| Utiliser des transactions | 70 |
| Lire un enregistrement | 71 |
| Impact sur le développement | 71 |
| Web services | 71 |
| Principes | 71 |
| REST | 71 |
| Activer les Web Services | 72 |
| Utilisation | 72 |
| Utiliser les web services | 72 |
| Accéder aux web services | 72 |

| | |
|--|----|
| Lire des données | 72 |
| Modifier des données..... | 73 |
| Supprimer des données..... | 73 |
| Ajouter des données | 74 |
| Les données..... | 74 |
| Les images | 74 |
| Les prix | 75 |
| Utilisation avancée | 75 |
| Filtrer et trier les informations..... | 75 |
| Limiter le nombre de zones à lire | 75 |
| Web service et ObjectModel..... | 76 |
| Gérer des liens entre différentes ressources..... | 76 |
| Gérer des liens entre différentes ressources..... | 76 |
| Définir le getter et le setter d'une ressource | 77 |
| Gérer l'ajout et la modification de ressources..... | 77 |
| Templating | 78 |
| Templating dans PrestaShop | 78 |
| Moteurs de templates | 78 |
| Smarty | 78 |
| Présentation | 78 |
| Smarty | 78 |
| Installation de Smarty | 78 |
| Utilisation de templates..... | 78 |
| Utilisation en PHP | 79 |
| Paramétrage de Smarty | 79 |
| Configuration | 79 |
| Configuration spécifique pour le front-office | 80 |
| Configuration spécifique pour le back-office..... | 80 |
| La compilation des templates..... | 80 |
| La mise en cache des templates..... | 80 |
| Bases syntaxiques | 81 |
| Délimiteurs..... | 81 |
| Les commentaires..... | 81 |
| Désactiver smarty | 81 |
| Les variables..... | 81 |
| Les conditions..... | 82 |
| Les boucles..... | 82 |
| Les captures | 82 |
| L'inclusion | 83 |

| | |
|--|----|
| L'héritage | 83 |
| Les variables dans Smarty | 83 |
| Utiliser les variables | 83 |
| Echappement..... | 84 |
| Les filtres | 84 |
| Créer une variable | 84 |
| La variable \$smarty..... | 84 |
| La variable \$link..... | 84 |
| La fonction \$link->getAdminLink() | 85 |
| Les filtres..... | 85 |
| Utiliser les filtres..... | 85 |
| Les filtres smarty..... | 86 |
| Les filtres PHP | 86 |
| lower | 86 |
| replace..... | 86 |
| capitalize..... | 87 |
| escape | 87 |
| truncate..... | 87 |
| json_encode | 88 |
| json_decode | 88 |
| cleanHtml..... | 88 |
| classname | 89 |
| classnames..... | 89 |
| convertAndFormatPrice | 90 |
| secureReferrer..... | 90 |
| Les fonctions | 90 |
| Utiliser les fonctions | 90 |
| Les fonctions natives | 90 |
| Les fonctions additionnelles | 91 |
| Les fonctions PHP | 91 |
| Les paramètres..... | 91 |
| La fonction url | 91 |
| La fonction render | 92 |
| La fonction form_field..... | 92 |
| La fonction hook | 92 |
| La fonction widget..... | 93 |
| La fonction widget_block..... | 93 |
| La fonction l | 93 |
| La fonction dateformat | 93 |

| | |
|--|-----|
| La fonction debug | 94 |
| Nouveautés de la 1.7..... | 94 |
| Filtres et fonctions supprimés | 94 |
| Echappement des variables..... | 94 |
| Les objets sont remplacés par des tableaux | 94 |
| L'héritage et les blocks..... | 95 |
| Mise en forme des informations par le code PHP | 96 |
| Presenter | 96 |
| Les classes 'Presenter' | 96 |
| Exercices..... | 97 |
| Héritage de template..... | 97 |
| Solution héritage de template | 97 |
| Modification d'une liste..... | 97 |
| Solution modification d'une liste | 97 |
| Twig | 98 |
| Twig..... | 98 |
| Moteur de template | 98 |
| Syntaxe de base | 98 |
| Les variables..... | 98 |
| Les filtres, syntaxe classique | 99 |
| La fonction filter | 99 |
| Sécurité - XSS | 99 |
| Les tests | 99 |
| Condition defined..... | 99 |
| Condition divisibleby | 99 |
| Condition empty..... | 100 |
| Les boucles..... | 100 |
| Les boucles avec un tableau..... | 100 |
| La variables loop..... | 100 |
| Les includes..... | 100 |
| Les macros | 100 |
| Les blocs | 101 |
| L'héritage..... | 101 |
| Les extensions ajoutées pour PrestaShop | 102 |
| Variables globales..... | 102 |
| Filtres (Twig_SimpleFilter)..... | 102 |
| Fonctions (Twig_SimpleFunction) | 102 |
| Mécanisme de surcharge..... | 102 |
| Module | 103 |

| | |
|--|-----|
| Organisation des modules | 103 |
| Module legacy | 103 |
| Module moderne | 103 |
| Fichiers de cache | 103 |
| Le fichier principal du module | 104 |
| L'interface WidgetInterface | 104 |
| Module legacy..... | 105 |
| Le répertoire classes | 105 |
| Le répertoire controllers..... | 105 |
| Le répertoire translations..... | 106 |
| Le répertoire upgrade | 106 |
| Le répertoire views | 106 |
| Création d'un module legacy | 107 |
| Création d'un module | 107 |
| Création du fichier d'accroche. | 107 |
| Options..... | 107 |
| Les différentes valeurs possibles pour tab..... | 107 |
| La fonction install() | 108 |
| La fonction uninstall() | 108 |
| WidgetInterface | 109 |
| Prendre en charge les Hooks | 109 |
| Le fichier template du module..... | 110 |
| Le fichier CSS de notre module | 110 |
| Ajouter une page de configuration | 111 |
| Afficher la page de configuration | 111 |
| Création d'un contrôleur back-office..... | 112 |
| Création d'un contrôleur back-office | 112 |
| Exemple de contrôleur back-office..... | 113 |
| La fonction d'installation | 113 |
| La fonction de désinstallation | 114 |
| Le fichier template du contrôleur back-office..... | 114 |
| Modification du module..... | 114 |
| Ajax..... | 115 |
| Principe d'Ajax..... | 115 |
| La fonction ajax de jQuery | 116 |
| Paramètres et fonctions..... | 116 |
| La fonction jQuery.load..... | 116 |
| La fonction jQuery.get..... | 116 |
| Valeur de retour sur un appel ajax avec jQuery | 117 |

| | |
|---|-----|
| Utilisation d'Ajax dans un Contrôleur back-office | 117 |
| Override de module | 117 |
| Override de module | 117 |
| Exemple d'override | 118 |
| Quand faire un override de module ? | 118 |
| Mise à jour de module | 118 |
| Principes | 118 |
| Mise à jour d'un module | 118 |
| Exemple de mise à jour de module | 119 |
| Sécurité | 120 |
| Règles élémentaires | 120 |
| Optimiser votre fichier .htaccess | 120 |
| Sécuriser votre installation | 120 |
| Manipulation des données | 120 |
| Never trust foreign data | 121 |
| Contrôler vos données | 121 |
| Injections SQL | 121 |
| SQL - Se protéger | 122 |
| XSS | 122 |
| XSS - Se protéger | 122 |
| CSRF | 123 |
| CSRF - Se protéger | 123 |
| Man in the middle | 124 |
| MITM Se protéger | 124 |
| Internationalisation | 125 |
| Principe de base | 125 |
| Contenu en plusieurs langues | 125 |
| Pack de localisation | 125 |
| Exemple de pack | 125 |
| Taxes | 126 |
| Transporteurs | 126 |
| Moyen de paiement | 126 |
| Traductions | 127 |
| Themes | 127 |
| Gestion des traductions | 127 |
| Nouveau système de traduction | 127 |
| Le domaine Shop | 127 |
| Le sous domaine Shop.Theme | 128 |
| Le sous domaine Shop.Demo | 128 |

| | |
|--|-----|
| Le sous domaine Shop.Notification | 128 |
| Le sous domaine Shop.Forms..... | 128 |
| Le domaine Admin..... | 128 |
| La gestion des devises..... | 129 |
| La classe Currency..... | 129 |
| Lire une devise | 129 |
| Convertir en devise | 129 |
| Convertir en devise (exemple)..... | 130 |
| Afficher une devise sur le back-office legacy | 130 |
| Afficher une devise sur le back-office legacy (exemple)..... | 131 |
| Afficher une devise sur le front-office..... | 131 |
| Afficher une devise avec Smarty..... | 131 |
| Afficher une devise avec taxe avec Smarty..... | 131 |
| Convertir une devise avec Smarty | 132 |
| Internationalisation des modules | 132 |
| Restrictions par pays | 132 |
| La traduction des modules de PrestaShop | 133 |
| Le domaine pour les modules..... | 133 |
| La traduction des modules (non PrestaShop)..... | 133 |

Développeur de module back-office

PrestaShop

PrestaShop

Prestashop est un CMS open Source dédié au e-commerce.

PrestaShop en 2017 :

- Plus de 250.000 boutiques en ligne.
- Une solution utilisée dans plus de 195 pays.
- Plus de 300 partenaires.
- Plus de 5000 modules et thèmes sur addons.
- Plus de 70 ambassadeurs dans le monde.

Liens connexes

Le site de PrestaShop (<https://www.prestashop.com/>)

Les dernières versions

- PrestaShop 1.7 (Novembre 2016)
- PrestaShop 1.6 (Mars 2014)
- PrestaShop 1.5 (Septembre 2012)

Numérotation de version

PrestaShop utilise SemVer depuis la 1.6.1.0

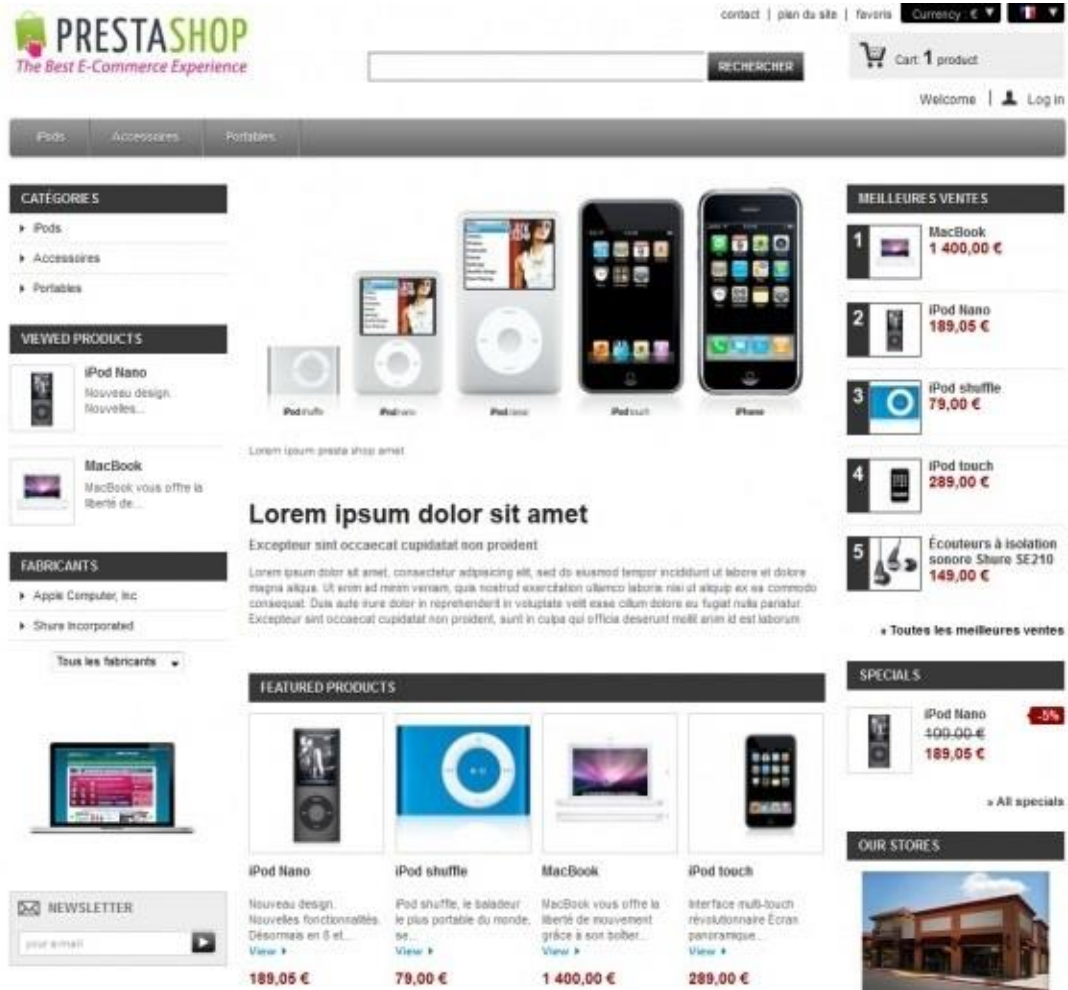
SemVer ou "Semantic Versioning" est une spécification du co-fondateur de Gravatar et ex CEO de GitHub, Tom Preston-Werner.

Il décrit son système de versioning comme suit :

Pour un numéro de version **1.MAJOR.MINOR.PATCH** : vous devez incrémenter :

- **MAJOR** : lorsque vous effectuez des modifications de l'API incompatibles
- **MINOR** : lorsque vous ajoutez des fonctionnalités de manière rétro-compatible
- **PATCH** : lorsque vous effectuez des corrections de bugs de compatibilité ascendante

Prestashop 1.5



Sortie en septembre 2012. (Réécriture complète du code PHP)

Technologies avec la 1.5

- PHP 5.2
- HTML 4 / CSS 2
- JavaScript + jQuery + jQuery mobile
- Smarty
- Thème "default" et Thème "mobile"
- Intégration de modules via hook
- Adaptation de la boutique via modules + overrides

Prestashop 1.6

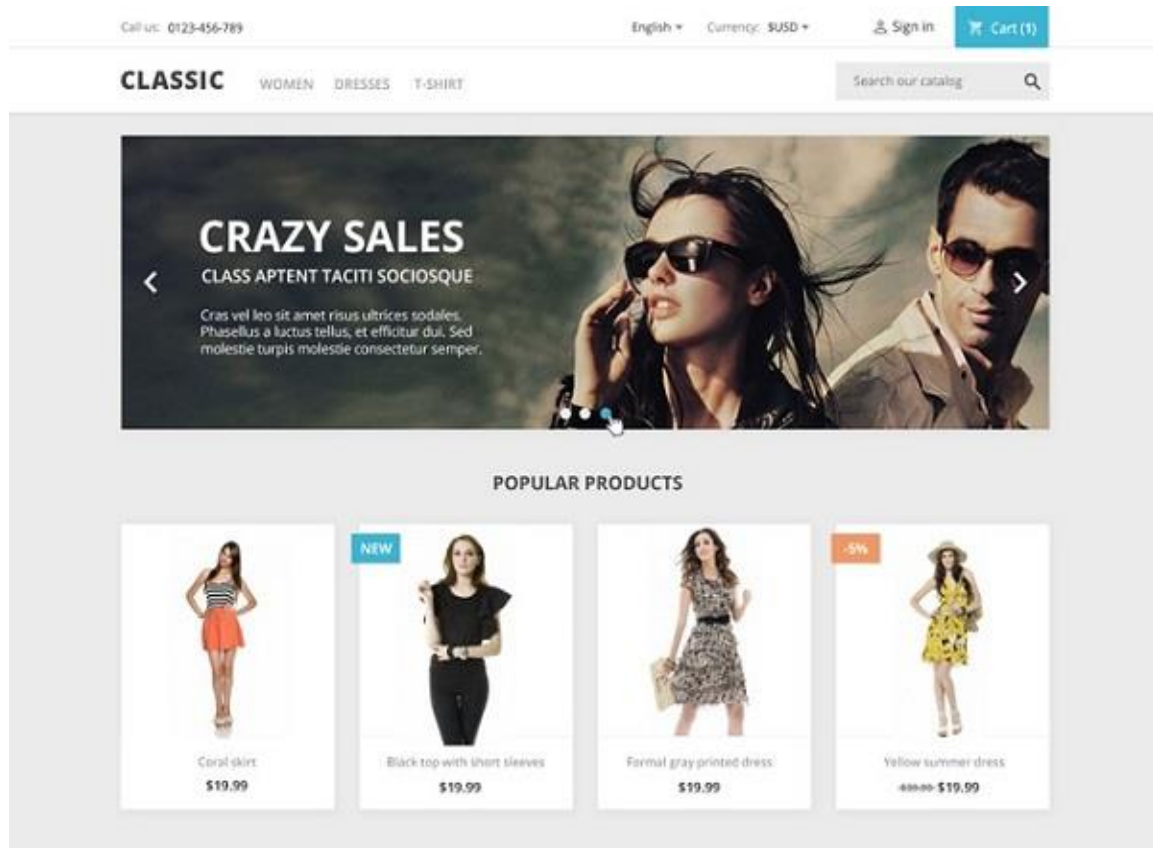


Sortie en mars 2014. (Réécriture des thèmes FRONT et BACK)

Technologies avec la 1.6

- PHP 5.2
- HTML 5 / CSS 3
- Bootstrap 3
- Sass + Compass
- JavaScript + jQuery - jQuery mobile
- Smarty
- Thème "default-bootstrap" - Thème "mobile"
- Intégration de modules via hook
- Adaptation de la boutique via modules + overrides

Prestashop 1.7



Sortie en novembre 2016. (Réécriture d'une partie du code PHP, Nouveau thèmes FRONT et BACK)

Technologies avec la 1.7

- **PHP 7.1 + Symfony 3 + Doctrine**
- **HTML 5 / CSS 3**
- **Bootstrap 3 et Bootstrap 4**
- **Sass + Compass + Bourbon (IE 9+)**
- **JavaScript + jQuery + ES 6 + Node.js + Webpack + Vue.js + Babel + ...**
- **Smarty + Twig**
- **Thème "classic" + "starter" (déprécié) + "classic-rocket"**
- **Intégration de modules via hook + widget**
- **Adaptation de la boutique via modules + overrides + bundle Symfony**

Les nouveautés de la 1.7

- Nouveaux thèmes : Starter Thème (déprécié), Classic, Classic rocket
- Refonte du back-office :
 - Menu / Navigation
 - Gestion des produits
 - Administration des modules
 - Gestion des thèmes
 - Gestion des traductions
 - Etc...

- Utilisation de Symfony2 puis Symfony3 avec Twig
- Nouveau tunnel de commande
- Nouvelle API pour les modules de paiement
- Nouvelle interface Widget pour les modules
- Nouveau système de traduction

Nouveautés en intégration

- Un Starter Thème (déprécié) pour créer des thèmes
- Le thème Classic rocket pour utiliser la dernière version de Bootstrap 4
- Une nouvelle architecture avec un fichier de configuration
- Des nouvelles fonctions pour Smarty
- Bootstrap 4, Material design, Font Material
- ES 6 pour le JavaScript
- Twig et un UI kit sur le back-office
- Une gestion des ressources avec Webpack

Nouveautés en développement

- PHP 5.4 (puis PHP 5.6 et PHP 7.1.3 depuis la version **1.7.7**)
- Nouvelle architecture CQRS / DDD (depuis la version **1.7.6**)
- Composer
- Dependency Injection / Factories
- Tests (PHPUnit/Selenium)
- CLDR (Unicode Common Locale Data Repository)
- Symfony 2 (Symfony 3 depuis la version 1.7.4)
- Utilisation d'interfaces, de services, de classes Presenter, Formatter, etc

Dernières suppressions

- Suppression de la comparaison des produits
- Suppression des scènes
- Suppression du live edit
- Suppression du ThemeConfigurator
- Suppression de la gestion des meta par thème.
- Suppression de la gestion de stock avancée

Nouvelles normes

- PHP : Symfony et PSR-2
- JavaScript : Airbnb JavaScript Style Guide
- CSS & HTML : Code Guide by @mdo (Mark Otto)

Liens connexes

[EN] Symfony Coding Standards

(<http://symfony.com/doc/current/contributing/code/standards.html>)

[EN] Airbnb JavaScript Style Guide (<https://github.com/airbnb/javascript>)

[FR] Airbnb JavaScript Style Guide (<https://github.com/nmussy/javascript-style-guide>)

[EN] Code Guide by @mdo (<http://codeguide.co/>)

[FR] Guide de Code HTML et CSS par @mdo (<http://pixelastic.github.io/code-guide/>)

Technologies utilisées par PrestaShop

Sass



Métalangage utilisé pour générer des feuilles de style.

- Compatible CSS 3
- Extension de langage permettant l'utilisation d'imbrications, de variables et de classes
- Nécessite un interpréteur pour générer des fichiers CSS
- Avec directives et instructions de contrôle comme (if, for, each, while, etc.)

Liens connexes

- [EN] Sass: Syntactically Awesome Style Sheets (<http://sass-lang.com/>)
- [FR] Sass Language ([https://fr.wikipedia.org/wiki/Sass_\(langage\)](https://fr.wikipedia.org/wiki/Sass_(langage)))
- [EN] Sass Language ([https://en.wikipedia.org/wiki/Sass_\(langage\)](https://en.wikipedia.org/wiki/Sass_(langage)))
- [EN] playground for Sass, Compass, and LibSass (<https://www.sassmeister.com/>)

Compass



Compass est un framework Sass Open Source.

- Un framework Sass
- Une gestion simplifiée des bibliothèques de style (Bootstrap)
- Une collection de mixins et de fonctions (Gestion des navigateurs, etc)
- Des outils pour générer automatiquement des sprites, etc

Liens connexes

- Le site de Compass (<http://compass-style.org/>)

Bourbon



Bourbon est une bibliothèque de mixin Sass.

C'est une librairie Sass qui n'est pas liée à Ruby comme Compass.

La bibliothèque Bourbon est utilisée, dans PrestaShop, pour assurer la compatibilité des thèmes avec IE9 (display: flex).

Liens connexes

- Le site de Bourbon (<http://bourbon.io/>)

Bootstrap 3



Bootstrap 3 est un framework qui facilite et accélère le développement. Il inclut :

- une base CSS (au format SASS) configurée à partir d'un fichier de variables
- des conventions de structure HTML et de nommage de classes
- une librairie javascripts pour les fonctions les plus courantes

Liens connexes

Le site de Bootstrap 3 (<http://getbootstrap.com/docs/3.3/>)

Bootstrap 4



Bootstrap 4 est une réécriture de *Bootstrap 3*, avec des modifications majeures.

- Fin du support pour IE8 et iOS 6. Désormais IE9+(flex) et iOS 7+
- Nouveau support pour Android v5.0, Lollipop's et WebView
- Abandon de LESS. Bootstrap 4 utilise uniquement du SCSS
- Passage de la taille de la font global de 14px à 16px
- Passage du px au rem (16px = 1rem)
- Nouvelle grille, ajout du XL
- Nouvelles variables de paramètres

Remarque : Bootstrap 4 (4.0.0-alpha.4) est REQUIS pour tous les thèmes PrestaShop 1.7.

Liens connexes

Le site de Bootstrap 4 (<https://v4-alpha.getbootstrap.com/>)

Smarty



Smarty est un moteur de rendu :

- Il facilite la séparation entre la logique applicative et la présentation.
- Propose une syntaxe propre et condensée dédiée au design.
- Fonctionnement par compilation puis cache.

Liens connexes

Smarty (<https://www.smarty.net/>)

Source de Smarty (<https://github.com/smarty-php/smarty/>)

Twig



Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le framework Symfony.

Symfony



Symfony

Symfony, une boîte à outils avec :

- une sélection de composants (les Composants Symfony) et des bibliothèques tierces (ex Swiftmailer pour envoyer des emails);
- une configuration et une bibliothèque « d'interface » qui lient toutes ces pièces ensembles.
- des conventions d'écriture et d'organisation

Outils

Composer



Composer est un gestionnaire de paquets et de dépendances pour vos projets en PHP. Il se charge de mettre à jour vos bibliothèques et de s'assurer qu'elles sont compatibles entre elles. Composer nécessite PHP 5.3.2 ou plus.

Liens connexes

Télécharger composer (<https://getcomposer.org/download/>)

Node.js



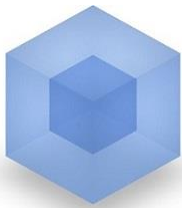
Node.js est un environnement de bas niveau permettant d'exécuter du JavaScript.

Pour installer Node.js, rendez-vous sur le site nodejs.org.

Liens connexes

Le site de nodejs (<https://nodejs.org/>)

Webpack



Webpack organise toutes les ressources (CSS, images, fontes, bibliothèques tierces) en module.

Avec *Webpack*, vous pouvez :

- Optimiser les ressources.
- Gérer les dépendances.
- Convertir les ressources (ES6 / ES5, etc.).
- Vérifier l'existence des ressources (images, fontes, etc.).

Git



Au même titre que *SVN* ou *CVS*, *GIT* est un protocole de gestion de version sous licence libre.

- Git est un système de contrôle de version libre et open source.
- Chaque clone Git est un dépôt du projet, avec l'historique et le suivi des révisions.
- Il ne dépend pas de l'accès au réseau ou à un serveur central.

Liens connexes

Msys Git (<https://code.google.com/p/msysgit/downloads/list>)

Tortoise GIT (<https://code.google.com/p/tortoisegit/wiki/Download?tm=2>)

GitHub



GitHub est un service en ligne qui permet d'héberger des dépôts de code.

Il assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme :

- Le suivi des bugs.
- Les demandes de fonctionnalités.
- La gestion de tâches.
- ...

Liens connexes

GitHub (<https://github.com/>)

PrestaShop sur GitHub (<https://github.com/PrestaShop/>)

Le validateur de modules

Le validateur a pour but de vous aider à soumettre des modules de très bonne qualité pour :

- Faciliter leurs compatibilités lors des évolutions du cœur de PrestaShop.
- Diminuer les demandes de support lors de l'intégration du module dans les boutiques existantes.
- Simplifier les mises à jour des boutiques qui utilisent votre module.
- ...

Liens connexes

Le validateur de modules (<https://validator.prestashop.com/>)

Le générateur de modules

PrestaShop PS GUIDE MODULE GENERATOR ACCOUNT

Generate a basic module

1 Basics 2 Settings 3 Hooks

Basics

Define the basics for your new generated module.

Module type: Generic module
Choose the module type you want to create.

Module tab: Administration
Choose the module type you want to create.

Module name: mymodule
Specify the name you want for your module (alpha-numeric characters only and 5 characters minimum).

Display name: My new module
Enter the name that will be display in the back-office (on the "Modules" page).

Description: Here is my new great module for PrestaShop!
Add a description for your module to help merchants to understand what he should use your module for (200 characters maximum).

Author: PrestaShop
Give your name or the name of your company.

Version: [dropdown] [dropdown] [dropdown]

Previous Next

Le générateur de modules permet de disposer en quelques clics d'un module respectant les normes de PrestaShop.

Liens connexes

Le générateur de modules (<https://validator.prestashop.com/generator>)

Documentation et Support

Documentation

Vous pouvez trouver plusieurs guides pour PrestaShop sur le site de la documentation (doc.prestashop.com).

Depuis la version 1.7, la documentation technique est disponible sur le site (devdocs.prestashop.com).

Liens connexes

PrestaShop user documentation (<http://doc.prestashop.com/>)

New PrestaShop 1.7 documentation (<http://developers.prestashop.com/>)

Development blog

Le blog est l'endroit où les équipes de PrestaShop communiquent directement avec la communauté.

Il est particulièrement destiné à ceux qui contribuent.

Liens connexes

PrestaShop development blog (<http://build.prestashop.com/>)

Les forums de discussion

Vous pouvez également trouver de nombreuses informations directement sur les forums de PrestaShop.

Liens connexes

Les forums de PrestaShop (<https://www.prestashop.com/forums/>)

Slack



PrestaShop n'utilise plus Gitter et propose de communiquer avec Slack.

Vous pouvez également retrouver sur Slack la communauté friends of presta.

Liens connexes

PrestaShop on Slack (https://join.slack.com/t/prestashop/shared_invite/zt-dkmbz5qf-I~FIEWwmRUOXunc5ui0Ucg)

Friends of Presta (Github) (<https://github.com/friends-of-presta>)

Friends of Presta (Slack) (https://join.slack.com/t/friends-of-presta/shared_invite/enQtNzQ5ODMyODY2MDgxLTA2MmE0MjQ5NTA3NjdmZWE5NzAyZDI2N2YzZTY1OGQwZDZhYmQyOTU0NGZhYmFmYmJkMGQxNzU2NzUzZjM2ZmM)

Installation et Configuration

Installation de PrestaShop

Les pré-requis

Pré-requis pour installer PrestaShop :

- Système : Unix, Linux, Mac OS ou Windows
- Serveur Web : Apache Web Server 2.2, Microsoft IIS 6.0 ou Nginx 1.0
- PHP : Version 7.1.3 ou supérieure
- Base de données : MariaDB, MySQL 5.5 ou plus
- Mémoire : Au moins 512Mo de RAM sur votre serveur

Liens connexes

PrestaShop system requirements checker (<https://github.com/PrestaShop/php-ps-info>)

Version de PHP

| PrestaShop | PHP |
|---------------|-------------|
| 1.5.x | 5.2 à 5.5 |
| 1.6.0.x | 5.2 à 5.5 |
| 1.6.1.x | 5.2 à 7.1 |
| 1.7.0 à 1.7.3 | 5.4 à 7.1 |
| 1.7.4 | 5.6 à 7.1 |
| 1.7.5 à 1.7.6 | 5.6 à 7.2 |
| 1.7.7 | 7.1.3 à 7.3 |

Liens connexes

[EN] Announcing End Of Support For Obsolete PHP Versions
(<https://build.prestashop.com/news/announcing-end-of-support-for-obsolete-php-versions/>)

Mettre en place un environnement local

Installer un serveur web (Apache), l'interpréteur du langage PHP , un serveur de base de données MySQL et idéalement l'outil d'administration phpMyAdmin.

Ils sont fréquemment regroupés dans des packages AMP : Apache + MySQL + PHP et le système d'exploitation, ce qui donne pour WAMP (Windows + Apache + MySQL + PHP), MAMP (Mac +...) et LAMP (Linux +...).

Étant donné que tous les éléments de ces packages sont open-source, ceux-ci sont la plupart du temps gratuits. Voici une sélection de quelques installateurs AMP gratuits :

- XAMPP (Windows, Mac, Linux, Solaris)
- WampServer (Windows)
- EasyPHP (Windows)
- MAMP (Mac)

Configurer PHP

Pour que PrestaShop fonctionne correctement, il faut que votre configuration de PHP dispose des réglages et bibliothèques suivants :

- MySQL
- Bibliothèque GD
- Extension Dom
- allow_url_fopen
- OpenSSL
- FileInfo
- Intl
- GZIP

Autres librairies recommandées : Mcrypt, Curl.

Installation avec git

Pour la version 1.6, vous devez utiliser l'option --recursive.

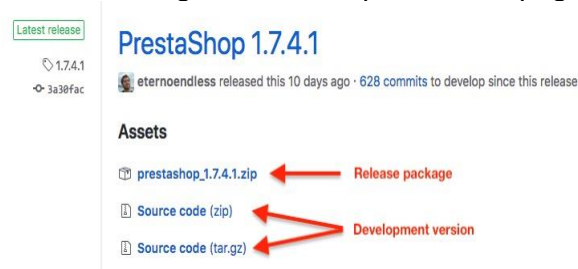
```
git clone https://github.com/PrestaShop/PrestaShop myfolder --recursive -b 1.6.1.x
```

Pour la version 1.7, les modules sont installés via composer.

```
git clone https://github.com/PrestaShop/PrestaShop myfolder -b 1.7.2.x
```

Installation par téléchargement

Pour télécharger PrestaShop, utiliser la page 'releases' sur GitHub.



A chaque version vous avez 3 fichiers :

- Le fichier de release : prestashop_1.MAJOR.MINOR.PATCH.zip
- Le fichier source au format zip
- Le fichier source au format tar.gz

Le fichier de release permet d'installer une boutique en mode production.

Le fichier source contient une copie du dépôt Git au moment de la publication de la version.

Liens connexes

PrestaShop sur GitHub (<https://github.com/PrestaShop/>)

Contenu du fichier release

Le fichier release contient 3 fichiers :

- **index.php**
- **Install_PrestaShop.html**

- **prestashop.zip**

Liens connexes

PrestaShop sur GitHub (<https://github.com/PrestaShop/>)

Autres techniques d'installation

Vous pouvez installer PrestaShop avec Docker ou Vagrant.



Vous pouvez également utiliser un fichier zip créé toutes les nuits à partir de GitHub.

Remarque : Vous trouverez des box Vagrant sur GitHub avec différentes configurations de serveur.

Liens connexes

PrestaShop Nightly Build (<https://nightly.prestashop.com/>)

PrestaShop on Docker (<https://hub.docker.com/r/prestashop/prestashop/>)

Créer une base de données.

Pour utiliser PrestaShop, vous devez disposer d'une base de données.

Vous pouvez créer la base de données pendant l'installation de PrestaShop ou avant de lancer l'installation.

Connectez-vous à PhpMyAdmin :

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

Utilisez la fonction "Créer une base de données" qui se trouve au centre de la page.

Installation à partir d'un navigateur.

PrestaShop a mis au point une procédure d'installation automatique depuis un navigateur.

Rendez-vous sur la page <http://127.0.0.1> et suivez les instructions.

Installation par ligne de commande.

Vous pouvez installer PrestaShop à partir d'une ligne de commande.

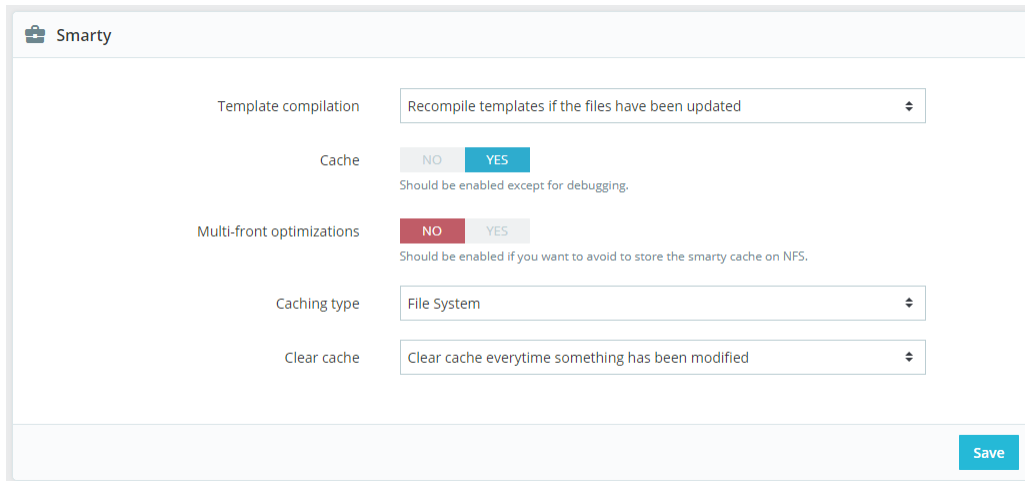
```
php install-dev/index_cli.php \
--domain=$domain \
--base_uri=/ \
--db_server=localhost \
--db_name=$project \
--db_user=$user \
--db_password=$pass \
--prefix=ps_ \
--email=$email \
--password=... \
--name=$project
```

Configuration

Compilation des templates Smarty

Lors du développement ou de la modification de templates Smarty (Fichier TPL de thèmes ou modules), vous devez **activer** la compilation pour visualiser vos modifications.

Rendez-vous dans le menu **Paramètres avancés > Performances**



The screenshot shows the 'Smarty' configuration panel. It includes the following settings:

- Template compilation:** A dropdown menu set to 'Recompile templates if the files have been updated'.
- Cache:** Radio buttons for 'NO' and 'YES'. 'YES' is selected. A note below says 'Should be enabled except for debugging.'
- Multi-front optimizations:** Radio buttons for 'NO' and 'YES'. 'NO' is selected. A note below says 'Should be enabled if you want to avoid to store the smarty cache on NFS.'
- Caching type:** A dropdown menu set to 'File System'.
- Clear cache:** A dropdown menu set to 'Clear cache everytime something has been modified'.
- A 'Save' button is located at the bottom right.

Activer l'affichage des erreurs.

Par défaut, PrestaShop n'affiche pas de message d'erreur.

Pour afficher les messages d'erreurs, vous devez modifier `_PS_MODE_DEV_` dans le fichier **/config/defines.inc.php**

```
<?php

/* Debug only */
if (!defined('_PS_MODE_DEV_')) {
    define('_PS_MODE_DEV_', false);
}
/* Compatibility warning */
define('_PS_DISPLAY_COMPATIBILITY_WARNING_', true);
```

Utiliser Nginx

Nginx n'utilise pas de fichier **.htaccess**

Vous devez configurer la réécriture des URL dans le fichier de configuration.

Consulter la documentation spécifique pour Nginx sur la documentation de PrestaShop.

Liens connexes

PrestaShop et Nginx

(https://github.com/PrestaShop/PrestaShop/blob/develop/docs/server_config/nginx.conf.dist)

Système de cache

PrestaShop propose en natif :

- Memcached
- APC

- Xcache

Pour activer l'utilisation du cache rendez-vous dans le menu **Paramètres avancés > Performances**

Environnement de travail

Installation des outils pour l'intégration et le développement.

Editeur

Pour travailler dans les meilleures conditions vous devez utiliser un éditeur "intelligent".

- Visual Studio Code
- Atom
- PHPStorm

Remarque : Vous devez installer les extensions pour JavaScript, Sass, Scss, PHP, Smarty, Twig, etc.

Composer



Composer est un gestionnaire de paquets et de dépendances pour vos projets en PHP. Il se charge de mettre à jour vos bibliothèques et de s'assurer qu'elles sont compatibles entre elles. Composer nécessite PHP 5.3.2 ou plus.

Liens connexes

Télécharger composer (<https://getcomposer.org/download/>)

Utiliser Composer

En combinant l'utilisation de Composer avec un outil de gestion de version comme Git, vous disposez des mêmes packages sur tous vos environnements (dev, test, prod, etc).

- **composer dump-autoload** : pour gérer l'autoload
- **composer install** : Pour installer les packages
- **composer update** : Pour mettre à jour les packages

Remarque : En cas d'installation avec git, vous devez lancer **composer install** dans le répertoire racine de votre installation.

Node.js



Node.js est un environnement de bas niveau permettant d'exécuter du JavaScript.

Pour installer Node.js, rendez-vous sur le site nodejs.org.

Liens connexes

Le site de nodejs (<https://nodejs.org/>)

Utiliser npm

PrestaShop utilise npm pour installer automatiquement les outils et ressources nécessaires aux thèmes.

- Rendez-vous dans le répertoire **/themes** . Exécutez : **npm install**
- Pour chaque thème, rendez-vous dans le répertoire **_dev** et exécutez : **npm install**

Remarque : Vous devez vous assurer de la présence du fichier **/_dev/package.json** dans le thème que vous voulez modifier.

Xdebug



Xdebug est une extension PHP.

Xdebug permet de personnaliser l'affichage des messages d'erreur PHP, de déboguer un script PHP, de générer des fichiers de trace pour le profiling...

Sans xdebug : Notice: Array to string conversion in C:\Web\presta-training\classes\Dispatcher.php on line 203

Avec xdebug :

| (!) Notice: Array to string conversion in C:\Web\presta-training\classes\Dispatcher.php on line 286 | | | | |
|---|--------|---------|------------------------------------|------------------------|
| Call Stack | | | | |
| # | Time | Memory | Function | Location |
| 1 | 0.0001 | 414168 | {main}() | ...index.php:0 |
| 2 | 0.0943 | 2790728 | Dispatcher->dispatch() | ...index.php:27 |
| 3 | 0.0943 | 2790728 | Dispatcher->getController() | ...Dispatcher.php:350 |
| 4 | 0.1000 | 2791248 | Dispatcher->useDefaultController() | ...Dispatcher.php:1078 |
| 5 | 0.1000 | 2791248 | Dispatcher->getDefaultController() | ...Dispatcher.php:339 |

Liens connexes

xdebug (<https://xdebug.org>)

Exercice



Installer un environnement de travail.

- Installer Composer
- Installer la dernière version de PrestaShop 1.7 (Fichier de release)
- Installer PrestaShop.

Exercice



Installer les outils pour modifier le thème.

- Installer Node.js (Version LTS).
- Installer les dépendances CSS, JS, etc (Via NPM).
- Compiler les ressources du thème.

Architecture

Environnements

Il existe 3 environnements dans PrestaShop :

- Le front-office.
- Le back-office legacy.
- Le back-office modern (Symfony).

Le front-office

L'environnement front-office :

- Bootstrap 4
- Smarty (Mode 1.7)
- CSS3 (Webpack)
- JavaScript ES6 (Webpack)

Le back-office legacy

L'environnement back-office legacy :

- Bootstrap 3
- Smarty (Mode 1.6)
- CSS 3 + Bourbon + Compass
- Font Awesome
- JavaScript ES5

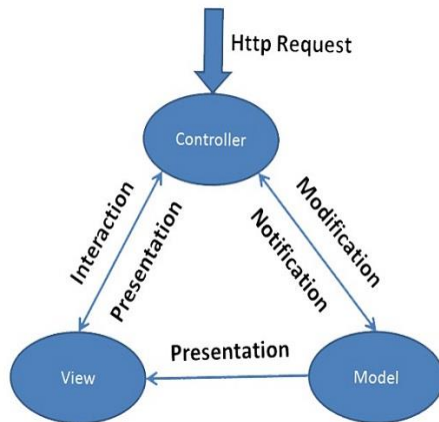
Le back-office modern

L'environnement back-office modern :

- Bootstrap 4
- Twig
- CSS3 (Webpack)
- Material icons
- JavaScript ES6 (Webpack)

Architecture de PrestaShop

Architecture MVC



Avantages :

- Il est plus facile de lire le code du logiciel.
- Les développeurs peuvent ajouter et corriger le code plus rapidement.
- Les designers et les intégrateurs peuvent travailler en toute sécurité dans le répertoire **/themes** sans avoir à comprendre ou même lire une seule ligne de code PHP.
- Les développeurs peuvent travailler sur des données et modules supplémentaires que les intégrateurs peuvent exploiter.

Modèles

- Le modèle représente le comportement de l'application : traitements des données, interactions avec la base de données, etc.
- Il décrit ou contient les données manipulées par l'application.
- Il assure la gestion de ces données et garantit leur intégrité.

Vues

- La vue correspond à l'interface avec laquelle l'utilisateur interagit.
- Sa première tâche est de présenter les résultats renvoyés par le modèle.
- Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, etc.).
- Ces différents événements sont envoyés au contrôleur. La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle et d'interagir avec l'utilisateur.

Contrôleurs

- Le contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser.
- Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer.
- Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle, et ce dernier notifie la vue que les données ont changées pour qu'elle se mette à jour.

Autoload

PrestaShop utilise un mécanisme d'autoload propriétaire pour le code legacy (sans namespace).

Depuis la version 1.7, Composer est utilisé pour le chargement des classes qui utilisent un namespace.

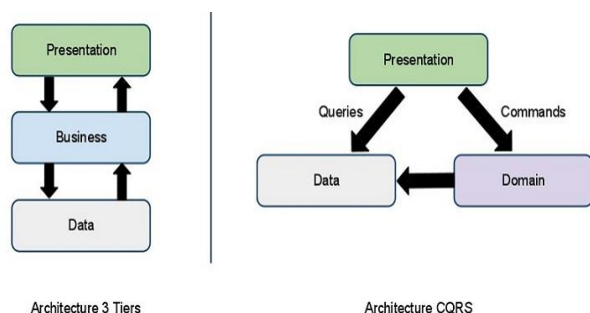
Stratégie de migration

Adapter pattern

- *Adapter* : permet d'utiliser le code legacy depuis le nouveau code de PrestaShop.
- *Core* : nouveau code de PrestaShop.
- *PrestaShopBundle* : nouveau back-office de PrestaShop.

Nouvelle architecture CQRS

A partir de PrestaShop **1.7.6** :



Command/Query Responsibility Segregation

Séparation des composants de traitement métier de l'information ("command" / écriture) et de restitution de l'information ("query" / lecture).

Avantages :

- Traitement asynchrone.
- Segmentation du code, création de micro-services.
- Maintenance et optimisation d'un micro-service plus simple.

Inconvénients :

- Le model CQRS augmente la complexité du code.
- Il faut assurer la cohérence entre la partie lecture et écriture du Model.

Nouvelle architecture DDD

A partir de PrestaShop **1.7.6** :

Approche décrite dans l'ouvrage **Domain Driven Design** d'Éric Evans.

Les objets et leurs interactions doivent être conçus de manière à représenter étroitement les concepts et les interactions de l'entreprise, plutôt que des abstractions techniques.

DDD permet de mieux appréhender la complexité d'un projet en partageant une approche et un langage communs par tous les membres impliqués dans la construction d'une application.

- Se concentrer sur le coeur du domaine.

- Explorer les modèles de son domaine via une collaboration créative des experts métiers et des développeurs.
- Parler un langage partagé (ubiquitous language) dans un contexte délimité (bounded context).

Pour en savoir plus : [/src/Core/Domain](#)

Liste des répertoires

Les répertoires de PrestaShop :

- /
 - **admin** : point d'entrée pour le back-office.
 - **app** : la configuration et les ressources de l'application.
 - **bin** : les fichiers exécutables du framework (vider le cache, maj de la bd, etc).
 - **cache** : les dossiers temporaires qui sont générés et réutilisés afin d'alléger la charge du serveur.
 - **classes** : les sources des modèles et des classes de base du Framework de PrestaShop.
 - **config** : les fichiers de configuration de PrestaShop.
 - **controllers** : les sources des contrôleurs de PrestaShop.
 - **docs** : de la documentation.
 - **download** : les produits numériques, pouvant être téléchargés : fichiers PDFs, MP3s, etc.
 - **img** : toutes les images de la boutique.
 - **install** : les sources de l'installateur de PrestaShop.
 - **js** : tous les fichiers JavaScript qui ne sont pas liés aux thèmes.
 - **localization** : tous les fichiers de localisation de PrestaShop.
 - **mails** : les modèles de mail.
 - **modules** : tous les modules de PrestaShop.
 - **override** : les fichiers qui supplantent les classes et contrôleurs par défaut de PrestaShop.
 - **pdf** : les modèles pour la génération PDF.
 - **src** : le nouveau code source de PrestaShop .
 - **themes** : les thèmes installés.
 - **tools** : les outils externes utilisés par PrestaShop.
 - **translations** : les traductions du coeur de PrestaShop.
 - **upload** : les fichiers en provenance des visiteurs de la boutique.
 - **var** : le cache et les logs de l'application.
 - **vendor** : les bibliothèques externes utilisées par PrestaShop.
 - **webservice** : les fichiers qui permettent aux applications de se connecter à PrestaShop via REST.

Répertoire modules

Le répertoire **/modules** :

- /
 - ...
 - mails
 - **modules**
 - **blockreassurance**
 - **contactform**

- **cronjobs**
- **dashactivity**
- ...
- override
- ...

Tous les modules sont installés dans le dossier **/modules**

Répertoire mails

Le répertoire **/mails** :

- **/**
 - ...
 - localization
 - **mails** : les modèles de mail
 - **en** : modèles en anglais
 - account.html
 - account.txt
 - backoffice_order.html
 - backoffice_order.txt
 - ...
 - **fr** : modèles en français
 - **themes** : thème d'email (PrestaShop 1.7.6+)
 - ...
 - modules
 - ...

Les modèles de mail sont dans le dossier **/mails**

Répertoire img

Le répertoire **/img** :

- **/**
 - ...
 - download
 - **img** : les images
 - **admin** : images utilisées par le panneau d'administration
 - **c** : images des catégories
 - **cms** : images chargées via l'éditeur Wysiwyg
 - **co** : images des attributs de type « couleur »
 - **l** : drapeaux utilisés par les langues
 - **m** : images des fabricants
 - **os** : images des statuts des commandes
 - **p** : images des produits
 - **s** : images des transporteurs
 - **st** : images des magasins
 - **su** : images des fournisseurs
 - **tmp** : dossier « temporaire »
 - ...
 - install

- ...

Toutes les images, gérées depuis le back-office de PrestaShop sont installées dans le dossier **/img**

Répertoire pdf

Le répertoire **/pdf** :

- **/**
 - ...
 - override
 - **pdf** : les modèles pour la génération des fichiers pdf.
 - .htaccess
 - delivery-slip.addresses-tab.tpl
 - delivery-slip.payment-tab.tpl
 - delivery-slip.product-tab.tpl
 - ...
 - src
 - ...

Tous les modèles de documents sont installés dans le dossier **/pdf**

Répertoire themes

Le répertoire **/themes** :

- **/**
 - ...
 - src
 - **themes** : les thèmes
 - _core
 - _libraries
 - **classic** : Le thème classic
 - core.js
 - core.js.map
 - debug.tpl
 - javascript.tpl
 - package.json
 - webpack.config.js
 - ...
 - tools
 - ...

Tous les thèmes sont installés dans le dossier **/themes**

Nouveautés de la 1.7.

Les nouveautés :

- Le répertoire **/css** à la racine de PrestaShop a été supprimé.
- Le répertoire **/log** à la racine de PrestaShop a été remplacé par le répertoire **/var/log**
- Les répertoires **/Core** et **/Adapter** ont été déplacés dans le répertoire **/src**

Méthodes

Dependency injection

Mécanisme permettant d'implémenter le principe de l'inversion de contrôle.

Disponible dans le constructeur de modules.

Cela permet de rendre les modules plus stables, car ils peuvent indiquer à l'application les composants dont ils ont besoin pour faire leur travail.

Exemple sans utilisation de "dependency" :

```
<?php

class MyModule extends Module
{
    public function __construct()
    {
        // initialization code
    }
    public function getSomeData()
    {
        // Where does Db come from?!
        Db::getInstance()->executeS('SELECT x FROM Y');
    }
}
```

Avec dependency injection

Exemple avec utilisation de "dependency" :

```
<?php

use Core\Foundation\Database;

class MyModule extends Module
{
    private $db;
    public function __construct(DatabaseInterface $db)
    {
        // initialization code
        $this->db = $db;
    }
    public function getSomeData()
    {
        $this->db->select('SELECT x FROM Y');
    }
}
```

Avantage :

- Pas d'accès à des variables globales.
- Plus facile à tester, utilisation d'objets simulés possible (mock).
- Dépendances évidentes lors de la lecture du code.
- Service Symfony.

Framework

La notion de contexte

Le contexte est une nouveauté technique de la version 1.5, il a été développé pour correspondre à deux objectifs :

- Ne plus utiliser des globales.
- Permettre de pouvoir changer le contexte dans certaines méthodes.

Le contexte est une légère implémentation du design pattern Registry, il s'agit d'une classe stockant les principales informations de PrestaShop comme le cookie, le customer, l'employé, le cart, smarty, etc..

Les objets stockés dans le contexte

Objets stockés dans le contexte :

- Language
- Country
- Currency
- Shop
- Cookie
- Link
- Smarty
- Customer
- Cart
- Controller
- Employee
- Translator (1.7)
- currentLocale (1.7.6)

Utilisation du context

Dans un contrôleur front-office et back-office legacy

```
// Get language id from context
$id_language = $this->context->language->id;
```

Dans une autre classe

```
// Get language id from context
$id_language = Context::getContext()->language->id;
```

Dans un module depuis PrestaShop 1.7.3

```
// @since 1.7.3
SymfonyContainer::getInstance()
    ->get('prestashop.adapter.legacy.context')
    ->language->id;
```


Les contrôleurs

Un contrôleur prend en charge la gestion des événements de synchronisation pour mettre à jour la vue ou le modèle et les synchroniser.

Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer.

Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle, et ce dernier notifie la vue que les données ont changées pour qu'elle se mettent à jour.

Tous les contrôleurs front-office et back-office legacy de Prestashop surchargent la classe *Controller*.

Pour le back-office (Code legacy) :

- *AdminController*
- *ModuleAdminController*

Pour le front-office :

- *FrontController*
- *ModuleFrontController*

Fonctionnement d'un contrôleur

Un contrôleur est lancé par le *Dispatcher*.

Les fonctions d'un contrôleur sont exécutées dans l'ordre suivant :

- `__construct()` : Sets all the controller's member variables.
- `init()` : Init controller.
- `setMedia()` : Adds all JavaScript and CSS specific to the page so it can be combined, compressed and cached.
- `postProcess()` process POST.
- `initHeader()` : Call before `initContent()`.
- `initContent()` : Init content.
- `initFooter()` : Call after `initContent()`.
- `display()` / `displayAjax()` / `displayAjaxAction()` : Ajax call.

La classe *ModuleFrontController*

Pour créer un nouveau contrôleur front-office dans un module, vous devez utiliser la classe *ModuleFrontController*.

Controller

.....|-- *FrontController*

.....|-- *ModuleFrontController*

La classe *ModuleAdminController*

Pour créer un nouveau contrôleur back-office legacy dans un module, vous devez utiliser la classe *ModuleFrontController*.

Controller

.....|-- *AdminController*

.....|-- *ModuleAdminController*

La surcharge d'un contrôleur.

Il vous est possible grâce à l'héritage de modifier ou d'ajouter de nouveaux comportements à un contrôleur front-office ou back-office legacy.

Les contrôleurs de PrestaShop sont dans le répertoire **/controllers** et utilisent le suffixe "Core".

Exemple pour le contrôleur *Category* :

- Fichier : **/controllers/front/listing/CategoryController.php**
- Classe : *CategoryControllerCore*

Pour surcharger un contrôleur, vous devez créer une nouvelle classe sans le suffixe "Core" et placer le fichier dans le dossier **/override/controllers/**

Exemple pour contrôleur *Category* :

- Fichier : **/override/controllers/front/listing/CategoryController.php**
- Classe : *CategoryController* extends *CategoryControllerCore*

Les vues

PrestaShop utilise des templates :

- *Smarty* pour la création des vues sur le front-office et le back-office legacy.
- *Twig* pour les vues du back-office moderne (Symfony).

Les templates Smarty

Les fichiers utilisés par le front-office sont dans le répertoire **/themes/{current_theme}/templates**

Les fichiers utilisés par le back-office legacy sont dans le répertoire **/admin-dev/themes/default/template**

Les templates Smarty utilisés sur le front-office

Le nom du fichier est généralement le même que celui du contrôleur qui l'utilise.

Exemple : Le contrôleur *ProductController* utilise la vue **/themes/{current_theme}/templates/catalog/product.tpl**

Remarque : Le nom du fichier est le nom du contrôleur converti en minuscule.

Les templates Smarty utilisés sur le back-office (Legacy)

Les contrôleurs back-office legacy utilisent des helpers.

Par défaut tous les helpers utilisent les mêmes templates.

Les templates des helpers peuvent être adaptés à chaque contrôleur. Les fichiers utilisés par un contrôleur sont regroupés dans un répertoire portant le même nom que le contrôleurs back-office.

Exemple : Le contrôleur *AdminDashboardController*

- Utilisez les fichiers de `//{admin-dev}/themes/default/template/controllers/dashboard`
- Utilisez les fichiers de `//{admin-dev}/themes/default/template/helpers`

Remarque : Le nom du répertoire est le nom du contrôleur converti du CamelCase au `underscore_case`.

La surcharge d'une vue sur le front-office

La notion de surcharge de vue pour le front-office n'existe pas.

Pour modifier une vue vous devez réécrire le template et le placer dans le répertoire de votre thème.

La surcharge d'une vue sur le back-office

Il est possible de surcharger les templates utilisés par un contrôleur back-office legacy.

Vous devez copier les templates du répertoire `//{admin-dev}/themes/default/template` vers le répertoire `/override/controllers/admin/templates`

Exemple pour le template utilisé par le contrôleur *AdminDashboard* :

- Copier le fichier `//{admin-dev}/themes/default/template/controllers/dashboard/helpers/view/view.tpl`
- Dans le fichier `/override/controllers/admin/templates/dashboard/helpers/view/view.tpl`

Les cookies

PrestaShop utilise des cookies cryptés pour stocker toutes les informations des sessions (Visiteurs / Clients et employés).

La classe *Cookie* (`/classes/Cookie.php`) est utilisée pour lire et écrire les cookies.

Utiliser les cookies dans PrestaShop

Depuis la version 1.5 de PrestaShop, vous devez utiliser le contexte.

Toutes les informations stockées dans les cookies sont ensuite disponibles grâce aux méthodes magiques `__get()` et `__set()` de PHP.

Exemples :

```
// Add my_custom_var value in cookie
Context::getContext()->cookie->my_custom_var = "the value"

// Check if value exist in cookie
if (isset(Context::getContext()->cookie->my_custom_var)) {
    $inCookieValue = Context::getContext()->cookie->my_custom_var;
}

// Remove value from cookie
unset(Context::getContext()->cookie->my_custom_var);
```

Informations stockées dans un Cookie Visiteur

Dans un Cookie Visiteur :

- *date_add* : The date and time the cookie was created (in YYYY-MM-DD HH:MM:SS format)

- *id_lang* : The ID of the selected language.
- *id_currency* : The ID of the selected currency.
- *last_visited_category* : The ID of the last visited category of product listings.
- *id_guest* : The guest ID of the visitor when not logged in.
- *id_connections* : The connection ID of the visitor's current session.
- *id_customer* : The customer ID of the visitor when logged in.
- *customer_lastname* : The last name of the customer.
- *customer_firstname* : The first name of the customer.
- *logged* : Whether the customer is logged in.
- *passwd* : The MD5 hash of the `_COOKIE_KEY_` in `config/settings.inc.php` and the password the customer used to log in.
- *email* : The email address that the customer used to log in.
- *id_cart* : The ID of the current cart displayed in the cart block.
- *id_address_invoice* : The ID of the current invoice address.
- *id_address_delivery* : The ID of the current delivery address.
- *checksum* : The Blowfish checksum used to determine whether the cookie has been modified by a third party. The customer will be logged out and the cookie deleted if the checksum doesn't match.

Informations stockées dans un Cookie Employé

Dans un Cookie employé :

- *date_add* : The date and time the cookie was created (in YYYY-MM-DD HH:MM:SS format).
- *id_lang* : The ID of the selected language.
- *id_employee* : The ID of the employee.
- *lastname* : The last name of the employee.
- *firstname* : The first name of the employee.
- *email* : The email address the employee used to log in.
- *profile* : The ID of the profile that determines which tabs the employee can access.
- *passwd* : The MD5 hash of the `_COOKIE_KEY_` in `config/settings.inc.php` and the password the employee used to log in.
- *checksum* : The Blowfish checksum used to determine whether the cookie has been modified by a third party. The employee will be logged out and the cookie deleted if the checksum doesn't match.

Utiliser les cookies - depuis un script indépendant de PrestaShop

Utilisez le code suivant :

```
include_once('path_to_prestashop/config/config.inc.php');
$cookie = new Cookie('ps-s1');
```

Vous devez déterminer le nom du cookie en fonction de la boutique.

Pour une boutique dans un groupe avec partage de commandes :

- `ps-sg` + l'id du groupe : Exemple "`ps-sg1`" pour le groupe de boutique 1

Pour une boutique indépendante :

- `ps-s` + l'id de la boutique : Exemple "`ps-s1`" pour la boutique 1

Remarque : Utilisez '`psAdmin`' pour lire le cookie d'un employé.

Classes du framework

PrestaShopAutoload

La classe *PrestaShopAutoload* permet le chargement automatique des classes de PrestaShop (depuis PrestaShop 1.5).

- Elle prend en charge la gestion des overrides.
- Elle utilise les fichiers **class_index.php**, **namespaced_class_stub.php** et **class_stub.php** du répertoire **/var/{env}/cache** de PrestaShop.

Dispatcher

La classe *Dispatcher* détermine le contrôleur à exécuter en fonction de l'url courante.

- La fonction *dispatch()* prend en charge l'analyse des routes et le lancement du contrôleur,
- Le hook *moduleRoutes* est le premier hook exécuté. Il permet le chargement des routes des modules,
- Le hook *actionDispatcherBefore* est traité avant l'instanciation du contrôleur,
- Le hook *actionDispatcher* est traité avant l'exécution du contrôleur,
- Le hook *actionDispatcherAfter* est traité après l'exécution du contrôleur.

ImageManager

La classe *ImageManager* permet de gérer les images de la boutique.

| Fonctions | Remarques |
|--------------------------------|---|
| thumbnail() | Generate a cached thumbnail for object lists (eg. carrier, order statuses...etc). |
| resize() | Resize, cut and optimize image. |
| isReallImage() | Check if file is a real image |
| isCorrectImageFileExt() | Check if image file extension is correct |
| validateUpload() | Validate image upload (check image type and weight) |
| cut() | Cut image |
| ... | ... |

Link

La classe *Link* permet de gérer les URL.

| Fonctions | Remarques |
|-------------------------------|---|
| getProductLink() | Create a link to a product |
| getRemoveFromCartURL() | Get the URL to remove the Product from the Cart |
| getUpQuantityCartURL() | Get URL to add one Product to Cart |
| getAddToCartURL() | Get add to Cart URL |

| Fonctions | Remarques |
|--------------------------|--------------------------------------|
| getCategoryLink() | Create a link to a category |
| getCMSLink() | Create a link to a CMS page |
| getModuleLink() | Create a link to a module controller |
| ... | ... |

Remarque : La classe *Link* est accessible via *\$link* dans Smarty.

Media

La classe *Media* permet de gérer les fichiers CSS et Javascript.

| Fonctions | Remarques |
|---------------------|---|
| packJS() | Minify JS |
| minifyCSS() | Minify CSS |
| cccCss() | Combine Compress and Cache CSS (ccc) |
| cccJS() | Combine Compress and Cache JavaScript (ccc) |
| clearCache() | Clear theme cache |
| addJsDef() | Add a new javascript definition at bottom of page (Smarty) |
| addJsDefL() | Add a new javascript definition from a capture at bottom of page (Smarty) |
| ... | ... |

PhpEncryption

La classe *PhpEncryption* prend en charge openssl 1.0.1+.

Fonctions principales : *encrypt(string)* et *decrypt(string)*

ProductAssembler

La classe *ProductAssembler* permet de compléter les informations disponibles dans la variable Smarty *\$product*

Depuis PrestaShop 1.7.0.

Tools

La classe *Tools* est la classe utilitaire du framework.

| Fonctions | Remarques |
|-------------------------|--------------------------------------|
| redirect() | Redirect user to another page |
| getServerName() | Get the server variable SERVER_NAME |
| secureReferrer() | Returns a safe URL referrer (smarty) |
| getValue() | Get a value from \$_POST / \$_GET |

| Fonctions | Remarques |
|-----------------------|--|
| getAllValues() | Get all values from \$_POST/\$_GET |
| getIsset() | Checks if a key exists either in \$_POST or \$_GET |
| safeOutput() | Sanitize a string |
| ... | ... |

Translate

La class *Translate* permet de gérer les traductions (Mode legacy).

Depuis PrestaShop 1.5.0.

Uploader

La classe *Uploader* permet de gérer le téléchargement de fichiers.

| Fonctions | Remarques |
|---------------------------|-----------------------------|
| setName() | Set file name |
| setSavePath() | Set file save path |
| process() | Upload and save file |
| checkUploadError() | Get translated upload error |
| ... | ... |

Validate

La classe *Validate* permet de valider les données.

| Fonctions | Remarques |
|-------------------------|--|
| isEmail() | Check for e-mail validity |
| isFloat() | Check for a float number validity |
| isImageSize() | Check for an image size validity |
| isCustomerName() | Check whether given customer name is valid |
| isModuleName() | Check for module name validity |
| isPrice() | Check for price validity |
| isAnything() | ... |
| ... | ... |

Les Helpers

Les Helpers pour le back-office legacy

Les Helpers (`/classes/helper`), vous permettent de générer des éléments HTML standard pour le back-office.

| Helper | Utilisation |
|--|---|
| HelperCalendar | dashboard |
| HelperForm | Générer un formulaire de saisie. |
| HelperImageUploader | Ancien contrôleur AdminProducts |
| HelperKpi et HelperKpiRow | Affichage d'indicateurs (Key Performance Indicator) |
| HelperList | Générer une liste d'éléments. |
| HelperOptions | Générer un formulaire de saisie pour les valeurs stockées dans la table de configuration. |
| HelperTreeCategorie et HelperTreeShops | Afficher l'arborescence des catégories et des boutiques |
| HelperUploader | Ancien contrôleur AdminProducts |
| HelperView | Afficher une page html. |

Les templates des Helpers

Les helpers utilisent des templates Smarty qui se trouvent dans le répertoire : `/admin-dev/themes/default/template/helpers/{helper_name}/`

Chaque modèle peut être surchargé dans le répertoire `/admin-dev/themes/default/controllers/{controller_name}/helpers/{helper_name}/`

Pour surcharger un template, il est préférable d'étendre le modèle parent, avec les balises `{block name="bloc_name"}`, et le mécanisme d'héritage de Smarty.

Exemple :

```
{extends file="helpers/view/view.tpl"}

{block name="override_tpl"}
{$smarty.block.parent}
<p>This page is generated by HelperView</p>
{/block}
```

Utiliser les Helpers

Vous pouvez utiliser les helpers dans vos modules pour la gestion de la page configuration, et dans les contrôleurs back-office legacy (*ModuleAdminController*).

Paramétrage des Helpers

Les helpers utilisent un tableau de paramètres.

Pour connaître les différents paramètres disponibles (en particulier pour le *HelperForm* ,

HelperOptions et le *HelperList*), vous pouvez consulter le contrôleur *AdminPatterns* (*/controllers/admin/AdminPatternsController.php*).

Nouveautés de la 1.7.

Espace de noms (namespaces)

PrestaShop 1.7 utilise des namespaces.

- Avec un namespace : il n'y a pas d'override possible.
- Sans namespace : l'override est possible mais déconseillé.

Espace PrestaShop\PrestaShop\Core

Réécriture des classes depuis la 1.6.1. :

- Test unitaire pour toutes les fonctions existantes dans Core.
- Pas de dépendance cachée (Dependency Injection).
- Pas d'accès direct ou indirect à une variable globale (*_GET*, *_POST*, ...)

Espace PrestaShop\PrestaShop\Adapter

Le namespace *PrestaShop\PrestaShop\Adapter* regroupe les classes permettant de passer du code legacy au nouveau code de PrestaShop.

Accéder au nouveau code depuis l'ancien

Depuis la version 1.7.1, accès direct.

Exemple dans la classe *Context* (legacy)

```
<?php
use PrestaShop\PrestaShop\Core\Localization\Locale;
use PrestaShopBundle\Translation\Loader\SqlTranslationLoader;
use PrestaShopBundle\Translation\TranslatorComponent as Translator;
use Symfony\Component\Filesystem\Filesystem;
use Symfony\Component\Finder\Finder;
use Symfony\Component\Translation\Loader\XliffFileLoader;
```

Dans la version 1.7.0, utilisation du *ServiceLocator*

Exemple dans la classe *Product* (legacy)

```
<?php
$stocks = \PrestaShop\PrestaShop\Adapter\ServiceLocator::get (
    '\\PrestaShop\\PrestaShop\\Core\\Foundation\\Database\\EntityManager')
->getRepository('Stock')
->findByIdProduct($this->id);
```

Remarque : Attention : *ServiceLocator* devrait disparaître. La fonction ne doit pas être utilisée dans un module.

Accéder à l'ancien code depuis le nouveau

Le nouveau code utilise la gestion des services Symfony et les classes du répertoire **/src/Adapter**

Exemple pour accéder aux devises : *prestashop.adapter.data_provider.currency* est déclaré comme service (fichier **service.yml**).

```
prestashop.adapter.data_provider.currency:  
    class: PrestaShop\PrestaShop\Adapter\Currency\CurrencyDataProvider
```

Le service est utilisé dans le nouveau contrôleur *SpecificPriceController* (**PrestaShopBundle\Controller\Admin\SpecificPriceController.php**) :

```
<?php  
  
$currencies = $this->container->get(  
    'prestashop.adapter.data_provider.currency'  
)->getCurrencies();
```

Tunnel de commande

Répertoire **/classes/checkout**

Nouvelles classes pour la gestion du tunnel de commande : *CheckoutProcess*

Classes utilisées par le contrôleur *OrderController* .

- Utilisation d'une interface pour chaque étape : *CheckoutStepInterface*
- Utilisation d'une classe de base pour chaque étape : *AbstractCheckoutStep*

Les différentes étapes :

- *CheckoutPersonalInformationStep*
- *CheckoutAddressesStep*
- *CheckoutDeliveryStep*
- *CheckoutPaymentStep*

Les classes utilitaires :

- *CartChecksum*
- *CheckoutSession*
- *ConditionsToApproveFinder* (terms of services, etc.)
- *DeliveryOptionsFinder* (Carrier list)
- *PaymentOptionsFinder* (New option in payment module)

Contrôleurs Front

Nouvelles classes dans le répertoire **/classes/controller** :

- *ProductPresentingFrontController* - classe abstraite pour les contrôleurs qui présente des produits.
- *ProductListingFrontController* - classe abstraite pour les contrôleurs qui présente des listes de produits.

Formulaires front-office

Nouvelles classes dans le répertoire **/classes/form**

Utilisation d'une interface pour les formulaires : *FormInterface*

```
<?php

interface FormInterface extends RenderableInterface
{
    public function setAction($action);
    public function fillWith(array $params = []);
    public function submit();
    public function getErrors();
    public function hasErrors();
    public function render(array $extraVariables = []);
    public function setTemplate($template);
}
```

Utilisation d'une interface pour la création des formulaires : *FormFormatterInterface*

Fonctionnement des formulaires sur front-office

Les formulaires sur le front-office utilisent 3 classes :

- *MyDataForm* : (Le contrôleur du formulaire) hérite de *AbstractForm* et implémente *FormInterface*
- *MyDataFormatter* : (La partie création des zones du formulaire) implémente *FormFormatterInterface*
- *MyDataPersister* : (La sauvegarde des informations).

Les formulaires existants sur le front-office

Les différents formulaires du front-office :

- *CustomerAddressForm*
- *CustomerForm*
- *CustomerLoginForm*

Les hooks disponibles dans les classes *Formatter* :

- *additionalCustomerAddressFields*
- *additionalCustomerFormFields*

API pour les modules de paiement

Suppression du hook *payment* et utilisation d'un tableau de paramètre pour l'affichage des moyens de paiement.

Possibilité de proposer le même moyen de paiement plusieurs fois.

Exemple de code :

```

<?php

public function hookPaymentOptions($params)
{
    if (!$this->active) {
        return;
    }
    if (!$this->checkCurrency($params['cart'])) {
        return;
    }
    $this->getTemplateVarInfos();

    $newOption = new PaymentOption();
    $newOption->setCallToActionText(
        $this->l('Pay by Bank Wire')
    )->setAction(
        $this->context->link->getModuleLink(
            $this->name, 'validation', array(), true
        )
    )->setAdditionalInformation(
        $this->context->smarty->fetch(
            'module:paymentexample/views/templates/front/payment_infos.tpl'
        )
    )->setLogo(
        Media::getMediaPath(_PS_MODULE_DIR_.$this->name.'/bankwire.jpg')
    );
    return [$newOption];
}

```

Exercice



Surcharge de la classe *FrontController*

Objectifs :

- Créer une nouvelle variable Smarty *\$currentController* disponible lors de l'affichage de toutes les pages.
- Cette variable permet de changer l'apparence de la page en fonction du contrôleur actif (Produit, Home, Page catégorie, etc.)

Remarque : Utiliser la fonction *get_class(\$this)* pour connaître le nom du contrôleur actif.

Solution



Créer le fichier **/override/classes/controller/FrontController.php**

```
<?php

class FrontController extends FrontControllerCore
{
    public function initHeader()
    {
        $this->context->smarty->assign(
            'currentController',
            get_class($this)
        );
        parent::initHeader();
    }
}
```

Dans le back-office, **Paramètres avancés > Performances** , Vider le cache

Modifier le fichier **/themes/classic/templates/_partials/header.tpl**

```
{* Add block on top of header.tpl *}
{block name="debug_output"}
    <div class="debug-output">
        Current controller : {$currentController}
    </div>
{/block}
```

Symfony

Répertoire de Symfony

Les répertoires :

- **app** : La configuration de l'application,
- **bin** : les scripts console,
- **src** : Le code PHP du projet,
- **vendor** : Les bibliothèques tierces.

Remarque : le répertoire **public** n'existe pas dans PrestaShop, il correspond à la racine de l'installation (**DOCUMENT_ROOT**).

Composants Symfony

Les composants :

- *HttpFoundation* : Contient les classes Request et Response, ainsi que d'autres classes pour la gestion des sessions, les uploads de fichiers, etc.
- *Routing* : Système qui permet de lier une URI spécifique à la fonction en charge de la gestion de l'action.
- *Form* : Une bibliothèque pour la création et la gestion de formulaires.
- *Validator* : Une bibliothèque pour créer et utiliser des règles de validation.
- *ClassLoader* : Une bibliothèque pour le chargement automatique « autoloader ».
- *Templating* : Une boîte à outils pour afficher des templates, gérer leur héritage, etc.
- *Security* : Une bibliothèque pour gérer tous les types de sécurité.
- *Translation* : Un framework pour traduire les chaînes de caractères.

Service container

Principe

Un *Service* est un objet qui est utilisé par votre application

Le *Container de services* permet de construire et de retrouver un objet

Il est possible de tagger un service pour l'associer à un traitement spécifique (twig.extension, console.command, ...).

Les services sont déclarés dans les fichiers **services.yml**

On peut surcharger chaque service existant, en utilisant un fichier nommé **config/services.yml** dans un module

Liens connexes

- [EN] Symfony services (PrestaShop)
(<https://devdocs.prestashop.com/1.7/modules/concepts/services/>)
- [EN] Service Container (https://symfony.com/doc/3.4/service_container.html)
- [EN] The DependencyInjection Component
(https://symfony.com/doc/3.4/components/dependency_injection.html)

Disponibilité

Sur toutes les pages du back-office.

Sur le front-office depuis la version **1.7.3**

Les services de Symfony

| Service | Fonction |
|-----------------------|---------------------------|
| twig | Twig - Moteur de template |
| database_connection | Doctrine (DBAL) |
| cache.app | Système de cache |
| filesystem et finder | Manipulation de fichier |
| authorization_checker | Utilitaire de sécurité |
| router | Gestion des URL |
| ... | ... |

Les services de PrestaShop

| Service | Fonction |
|---|----------------------------------|
| prestashop.adapter.cache_clearer | Vide les caches de PrestaShop |
| prestashop.core.api.stock_movement.repository | Le dépôt des mouvements de stock |
| prestashop.utils.zip_manager | Gestionnaire de fichiers Zip |
| ... | ... |

Pour afficher tous les services de PrestaShop :

```
php bin/console debug:container | grep 'prestashop'
```

La console Symfony

Principe

La console de Symfony permet d'interagir avec une application en ligne de commandes.

La console est un script PHP du répertoire **/bin** .

Il est possible de connaître la liste des commandes disponibles en exécutant la console sans paramètres.

```
php bin/console
```

Les commandes PrestaShop

PrestaShop propose de nouvelles commandes disponibles depuis la console Symfony :

- *prestashop:licenses:update* Rewrite your licenses to be up-to-date.

- *prestashop:mail:generate* Generate mail templates for a specified theme.
- *prestashop:module* Manage your modules via command line.
- *prestashop:schema:update-without-foreign* Update the database.
- *prestashop:taxes:update-eu-tax-rule-groups* Update EU Tax rule groups.
- *prestashop:theme:enable* Manage your themes via command line.
- *prestashop:theme:export* Create zip to distribute theme with its dependencies.
- *prestashop:translation:find-duplicates* Find duplicates of your translations.

Il est possible de connaître la liste des commandes PrestaShop disponibles :

```
php bin/console | grep 'prestashop'
```

La commande *prestashop:module*

La commande *prestashop:module* permet de gérer les modules en ligne de commande.

Les actions possibles :

- *install* pour installer un module.
- *uninstall* pour désinstaller un module.
- *enable* pour activer un module.
- *disable* pour désactiver un module.
- *enable_mobile* pour activer un module sur mobile.
- *disable_mobile* pour désactiver un module sur mobile.
- *reset* pour réinstaller un module.
- *upgrade* pour mettre à jour un module.
- *configure* pour configurer un module.

Exemple d'utilisation :

```
php bin/console prestashop:module disable ps_banner
```

La commande *cache:clear*

La commande *cache:clear* permet de vider le cache de PrestaShop.

```
php bin/console cache:clear
```


Data Access

Infrastructure

Système de gestion de base de données (SGBD)



PrestaShop supporte *MySQL* à partir de la version 5.5.3 et *MariaDB* à partir de la version 5.5.



MariaDB est souvent plus rapide que *MySQL*.

Configuration recommandée

Configuration du cache :

- `query_cache_limit = 128K`
- `query_cache_size = 32M`
- `query_cache_type = ON`

Optimiser MySQL / MariaDB

PrestaShop recommande la configuration suivante :

- `table_open_cache = 1000`
- `read_buffer_size = 2M`
- `read_rnd_buffer_size = 1M`
- `thread_cache_size = 80`
- `join_buffer_size = 2M`
- `sort_buffer_size = 2M`
- `max_connections = 400`
- `tmp_table_size = 32M`
- `max_heap_table_size = 32M`
- `table_definition_cache = 1000`
- `performance_schema = OFF`

La valeur du paramètre `innodb_buffer_pool_size` doit être supérieure à la place occupée par la base de données sur le serveur.

Après une mise à jour importante de la base de données, vous pouvez lancer une vérification de la base de données pour réoptimiser les indexes.

```
mysqlcheck -a -A -uroot -pyour_password
```

Plusieurs bases de données

Vous pouvez configurer PrestaShop pour utiliser plusieurs bases de données.

PrestaShop reconnaît une base de données "Master" pour les requêtes d'écriture. Et une ou plusieurs bases de données "Slave" pour les requêtes de lectures.

La base de données "Master" est configurée dans le fichier `/app/config/parameters.php`

La ou les bases de données "Slave" sont configurées dans le fichier `/config/db_slave_server.inc.php`

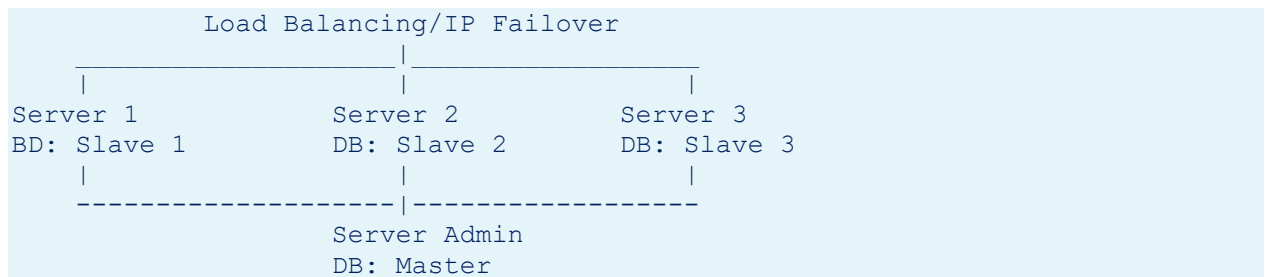
Remarque : Pour utiliser une base de données "Slave" : le paramètre `_PS_USE_SQL_SLAVE_` doit être précisé lors de l'appel à la fonction `Db::getInstance`.

Exemple d'infrastructure

Plusieurs bases de données pour distribuer les requêtes de lecture.

L'utilisation de plusieurs bases de données "Slave" permet de configurer plusieurs serveurs "Front"

Exemple de configuration :

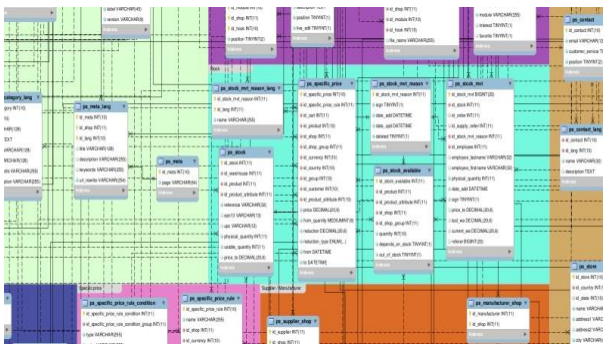


Les requêtes de lectures sur les différents serveurs sont exécutées en local.

Les requêtes d'écritures sont toutes exécutées sur la base Master.

Schéma de la base de données

Diagramme



Le diagramme de la base de données est disponible en téléchargement.

Liens connexes

Database diagram (PNG 2.75MB)

(<http://docs.prestashop.com/download/attachments/9961497/pdm-1.5.png>)

Database diagram (MySQL Workbench)

(<http://docs.prestashop.com/download/attachments/9961497/pdm-1.5.mwb>)

MySQL Workbench (Téléchargement) (<http://wb.mysql.com/>)

Encodage des caractères

Jusqu'à la version 1.7.6 de PrestaShop, l'encodage utilisé est *utf8*

A partir de PrestaShop 1.7.7, l'encodage utilisé est *utf8mb4* (Ajout du support des emojis, etc.).

Le préfixe des tables

Un préfixe (*_DB_PREFIX_*) est ajoutée devant les noms des tables utilisées par PrestaShop.

Ce préfixe permet d'exécuter plusieurs installations de PrestaShop avec une base de données unique.

Il est recommandé de changer le préfix par défaut (*ps_*) pour mieux se protéger des injections SQL.

Le moteur de stockage

Le moteur de stockage est utilisés par le système de gestion de base de données (*MySQL* / *MariaDB*) pour enregistrer et lire les données.

PrestaShop utilise par défaut le moteur *InnoDB*

Vous pouvez choisir un autre moteur (comme *MyISAM*) en modifiant le paramètre *database_engine* dans le fichier [/app/config/parameters.php](#)

Remarque : Depuis PrestaShop 1.7, le changement de moteur n'est plus proposé dans la procédure d'installation.

Choisir un moteur de stockage

Depuis PrestaShop 1.7, il est recommandé d'utiliser le moteur *InnoDB* :

- *InnoDB* permet de créer des contraintes d'intégrité (Nécessaire pour *Doctrine*).
- *MyISAM* peut être plus rapide (en écriture) et est compatible avec les sauvegardes "snapshot" d'OVH.

Création de la base de données

La base de données est créée lors de l'installation de PrestaShop.

Le schéma de la base est décrit dans le fichier [/install-dev/data/db_structure.sql](#)

Liens connexes

Database schéma (GitHub) (https://github.com/PrestaShop/PrestaShop/blob/develop/install-dev/data/db_structure.sql)

Mise à jour de la base de données

La base de données peut être mise à jour avec le script [/install-dev/upgrade/upgrade.php](#)

Ce script détermine les fichiers de mises à jour qui doivent être exécutés sur la base de données.

Les fichiers de mise à jour sont dans le répertoire [/install-dev/upgrade/sql](#)

Les fichiers de mise à jour peuvent lancer des scripts PHP du répertoire [/install-dev/upgrade.php](#)

Remarque : Le script `upgrade.php` est utilisé par le module "1-click upgrade".

Conventions de nommage

Normes pour le nom des tables

Le nom des tables :

- Le nom d'une table correspond au nom de l'entité qu'elle contient.
- Les noms de tables utilisent un préfixe 'ps_' qui peut être modifié lors de l'installation de PrestaShop.
- Les noms de tables sont en minuscules et utilisent '_' pour séparer les mots.
- Lorsqu'une table contient des liens entre deux entités, le nom des deux entités est précisé dans le nom de la table.
- Exemple : `ps_category_product` permet de relier les produits aux catégories.

Suffixe `_lang`

Les tables `xxxx_lang` :

- Les tables qui utilisent le suffixe `_lang` contiennent les traductions.
- Exemple : `ps_product_lang` contient les traductions pour la table `ps_product`.

Suffixe `_shop`

Les tables `xxxx_shop` :

- Les tables qui utilisent le suffixe `_shop` contiennent les enregistrements permettant de faire le lien avec une boutique.
- Exemple : `ps_category_shop` contient la position de chaque catégorie en fonction de la boutique.

Normes pour le nom des champs

Le nom d'un champ :

- Le nom d'un champ correspond au nom de l'information qu'il contient
- Les noms des champs sont en minuscules et utilisent '_' pour séparer les mots
- La clé primaire d'une table est toujours : `id_nomdelatable`. Exemple : `id_product`

`id_lang` et `id_shop`

Clé primaire des table Lang et Shop :

- Utilisez le champ `id_lang` pour préciser la langue associée à un enregistrement
- Utilisez le champ `id_shop` pour préciser la boutique associée à un enregistrement

DB

La classe DB

La classe `DB` (abstract) permet de lancer des requêtes SQL. La méthode `getClass()` renvoie une instance de la classe `DbPDO` ou `DbMySQLi` en fonction des librairies disponibles.

Les classes sont dans le répertoire **/classes/db**

Caractéristiques :

- Singleton,
- Peut gérer plusieurs bases de données,
- Peut utiliser un système de cache,
- Gestion automatique des bases de données répliquées (*_PS_USE_SQL_SLAVE_*),
- Affichage des messages d'erreurs uniquement si *_PS_DEBUG_SQL_* est à *true*

Gestion du cache

Par défaut le cache est activé en fonction de la valeur de *_PS_CACHE_ENABLED_*

Le mécanisme de cache permet de stocker les valeurs de retour des requêtes SQL (Le cache utilise le nom des tables dans la clé de stockage...). Vous pouvez configurer le système de cache dans le back-office, **Paramètres avancés > Performances**

| Fonctions | Remarques |
|-----------------------|---------------------------|
| disableCache() | Pour désactiver le cache. |
| enableCache() | Pour activer le cache. |

Fonction getInstance()

getInstance(\$master = true) permet d'accéder à la base de données "Master" ou à la prochaine base de données "Slave"

Remarque : La ou les bases de données "Slave" sont configurées dans le fichier **/config/db_slave_server.inc.php**

Fonction query()

query(\$sql) permet d'exécuter une requête SQL et de traiter le résultat.

Valeur possible de retour : *bool|mysqli_result|PDOStatement|resource*

Exemple d'utilisation :

```
<?php

function dumpCustomer()
{
    $data = Db::getInstance()->query(
        'SELECT * FROM `'. _DB_PREFIX_ .'customer`'
    );
    $sizeof = Db::getInstance()->numRows();

    if ($data && $sizeof > 0) {
        while ($row = Db::getInstance()->nextRow($data)) {
            foreach ($row as $field => $value) {
                echo Tools::safeOutput($field.'='.$value."\n");
            }
        }
    }
}
```

Fonction insert()

insert(\$table, \$data,...) permet de créer automatiquement une requête d'insertion dans une table à partir du tableau (key, value) ou d'un tableau de tableaux pour gérer plusieurs enregistrements.

Exemple d'utilisation :

```
<?php

Db::getInstance()->insert(
    'category_group',
    [
        ['id_category' => 1, 'id_group' => 1],
        ['id_category' => 1, 'id_group' => 2],
    ]
);
```

Remarque : En fonction des paramètres, la requête peut utiliser *INSERT* , *INSERT IGNORE* ou *REPLACE* .

Fonction update()

update(\$table, \$data, \$where = ",...) permet de créer automatiquement une requête de mise à jour dans une table à partir du tableau (key, value) ou d'un tableau de tableaux pour gérer plusieurs enregistrements.

Exemple d'utilisation :

```
<?php

Db::getInstance()->update(
    'customer',
    [
        'firstname' => 'Pierre',
        'lastname' => 'DURAND'
    ],
    'id_customer = 1'
);
```

Fonction delete()

delete(\$table, \$where = ",...) permet de créer automatiquement une requête de suppression dans une table.

Exemple d'utilisation :

```
<?php

function cleanCategoryGroupByCategory($id_category)
{
    return Db::getInstance()->delete(
        'category_group',
        'id_category = ' . (int)$id_category
    );
}
```

Fonction execute()

execute(\$sql, \$use_cache = true) permet d'exécuter une requête qui ne retourne pas

d'enregistrement.

Exemple d'utilisation :

```
<?php

function removeMyDataTableFromDb()
{
    return Db::getInstance()->execute(
        'DROP TABLE IF EXISTS `._DB_PREFIX_`mydata'
    );
}
```

Fonction executeS()

executeS(\$sql, \$array = true, \$use_cache = true) permet d'exécuter une requête qui retourne des enregistrements.

Exemple d'utilisation :

```
<?php

function getAllImages()
{
    return Db::getInstance()->executeS('
        SELECT `id_image`, `id_product`
        FROM `.`._DB_PREFIX_`.image`
        ORDER BY `id_image` ASC'
    );
}
```

Remarque : La fonction retourne un tableau des enregistrements ou *false* si aucun enregistrement n'est trouvé.

Fonction getRow()

getRow(\$sql, \$use_cache = true) permet d'exécuter une requête qui retourne UN enregistrement.

Exemple d'utilisation :

```
<?php

function getCurrencyById($id_currency)
{
    return Db::getInstance()->getRow(
        'SELECT `id_currency` FROM `._DB_PREFIX_`.currency'
        .' WHERE `id_currency` != ' .(int)$id_currency
        .' AND `deleted` = 0'
    );
}
```

Remarque : La fonction ajoute automatiquement LIMIT 1 à la requête et retourne l'enregistrement ou *false* si aucun enregistrement n'est trouvé.

Fonction getValue()

getValue(\$sql, \$use_cache = true) permet d'exécuter une requête qui retourne UNE valeur.

Exemple d'utilisation :

```
<?php

function getDeliveryNumber($order_invoice_id)
{
    return Db::getInstance()->getValue(
        'SELECT `delivery_number`
        FROM `'. _DB_PREFIX_ . 'order_invoice`
        WHERE `id_order_invoice` = '.(int)$order_invoice_id
    );
}
```

Remarque : La fonction retourne la valeur de la première colonne du premier enregistrement ou *false* si aucun enregistrement n'est trouvé.

Fonction numRows()

numRows() permet de connaître le nombre d'enregistrements retournés par la dernière requête.

Exemple d'utilisation :

```
<?php

function dumpCustomer()
{
    $data = Db::getInstance()->query(
        'SELECT * FROM `'. _DB_PREFIX_ . 'customer`'
    );
    $sizeof = Db::getInstance()->numRows();

    if ($data && $sizeof > 0) {
        while ($row = Db::getInstance()->nextRow($data)) {
            foreach ($row as $field => $value) {
                echo Tools::safeOutput($field.'='.$value."\n");
            }
        }
    }
}
```

Fonction escape()

escape(\$string, \$html_ok = false, \$bq_sql = false) permet de protéger les caractères spéciaux (NULL (ASCII 0), \n, \r, \, ', ", et Control-Z) d'une chaîne pour l'utiliser dans une requête SQL.

Exemple d'utilisation :

```
<?php

function removeRequiredFieldByObjectName($object_name)
{
    Db::getInstance()->execute(
        'DELETE FROM `'. _DB_PREFIX_ . 'required_field'
        .' WHERE object_name = "'.Db::getInstance()->escape($object_name).'"'
    );
}
```

Remarque : L'alias *pSQL()* permet également d'exécuter cette fonction.

Les fonctions d'interface avec PDO et MySQLi

| Fonctions | Remarques |
|-------------------------|---|
| connect() | Tries to connect to the database |
| createDatabase() | Tries to connect and create a new database |
| disconnect() | Destroys the database connection link |
| _query() | Executes an SQL statement, returning a result set or true/false |
| nextRow() | Returns the next row from the result set |
| getAll() | Returns all rows from the result set |
| _numRows(...) | Returns row count from the result set |
| Insert_ID() | Returns ID of the last inserted row |
| Affected_Rows() | Return the number of rows affected by the last SQL query |
| getMsgError() | Returns error message |
| getNumberError() | Returns error code |
| getVersion() | Returns database server version |
| _escape() | Escapes illegal characters in a string |
| set_db() | Switches to a different database |
| getBestEngine() | Selects best table engine (MyISAM or InnoDB) |

Autres fonctions

| Fonctions | Remarques |
|---------------------------------|--|
| displayError() | Displays last SQL error |
| checkConnection() | Try a connection to the database |
| checkEncoding() | Try a connection to the database and set names to UTF-8 |
| hasTableWithSamePrefix() | Try a connection to the database and check if at least one table with same prefix exists |
| checkCreatePrivilege() | Tries to connect to the database and create a table (checking creation privileges) |
| checkAutoIncrement() | Checks if auto increment value and offset is 1 |
| getLink() | Get used link instance |
| ... | ... |

ObjectModel

La Classe ObjectModel

C'est la classe principale du modèle objet de PrestaShop. La classe est défini dans le fichier **/classes/ObjectModel.php**

Remarque : Surcharge possible mais non conseillée.

Classe de type Active Record

Les attributs d'une table sont encapsulés dans la classe (cf variable static *\$definition*).
L'instance de la classe, est lié à un enregistrement de la base.

Utilisation

Exemple :

```
// Create New Zone
$obj = new Zone();
$obj->name = 'My Zone';
$obj->active = true;
$obj->save();

// Disabled Zone id 5
$obj = new Zone(5);
$obj->active = false;
$obj->save();

// Delete Zone id 14
$obj = new Zone(14);
$obj->delete();
```

La définition du modèle

Vous devez utiliser la variable static *\$definition* pour définir le modèle.

Exemple :

```

/**
 * @see ObjectModel::$definition
 */
public static $definition = [
    'table' => 'supplier',
    'primary' => 'id_supplier',
    'multilang' => true,
    'fields' => [
        'name' => ['type' => self::TYPE_STRING, 'validate' => 'isCatalogName'
, 'required' => true, 'size' => 64],
        'active' => ['type' => self::TYPE_BOOL],
        'date_add' => ['type' => self::TYPE_DATE, 'validate' => 'isDate'],
        'date_upd' => ['type' => self::TYPE_DATE, 'validate' => 'isDate'],

        /* Lang fields */
        'description' => ['type' => self::TYPE_HTML, 'lang' => true, 'validat
e' => 'isCleanHtml'],
        'meta_title' => ['type' => self::TYPE_STRING, 'lang' => true, 'valida
te' => 'isGenericName', 'size' => 128],
        'meta_description' => ['type' => self::TYPE_STRING, 'lang' => true, '
validate' => 'isGenericName', 'size' => 255],
        'meta_keywords' => ['type' => self::TYPE_STRING, 'lang' => true, 'val
idate' => 'isGenericName', 'size' => 255],
    ],
];

```

Les différents types de zone

Les types sont utilisés pour sécuriser l'insertion des données dans les tables.

Valeurs autorisées en fonction du type :

- **TYPE_BOOL** : 0 ou 1,
- **TYPE_DATE** : force 0000-00-00 si la zone est vide,
- **TYPE_FLOAT** : valeur flottante et remplace automatiquement ',' par '.',
- **TYPE_HTML** : chaîne de caractère avec html (Utilisation de *HTMLPurifier*),
- **TYPE_INT** : integer,
- **TYPE_NOTHING** : aucun contrôle, toutes les valeurs sont acceptées.
- **TYPE_STRING** : chaîne de caractère sans html (*strip_tags* automatique),
- **TYPE_SQL** : chaîne de caractère sans contrôle.

Pour un objet en plusieurs langues

Il faut ajouter dans le tableau de définition '*multilang*' => *true* , et ajouter dans la liste des champs '*lang*' => *true* pour préciser que le champ est traduit.

Pour un objet en fonction de la boutique et en plusieurs langues

Il faut ajouter dans le tableau de définition '*multilang_shop*' => *true*

DBQuery

La classe DBQuery

Un query builder pour simplifier la création des requêtes SQL.

Exemple d'utilisation :

```
<?php

$sql = new DbQuery();
$sql->select('*');
$sql->from('cms', 'c');
$sql-
>innerJoin('cms_lang', 'l', 'c.id_cms = l.id_cms AND l.id_lang = '.(int)$id_l
ang);
$sql->where('c.active = 1');
$sql->orderBy('position');
return Db::getInstance()->executeS($sql);
```

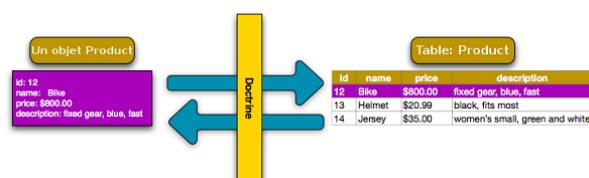
Les méthodes

| Fonctions | Remarques |
|--------------------|--|
| build() | Generate and get the query (use __toString()) |
| from() | Set table for FROM clause |
| groupBy() | Add a GROUP BY restriction |
| having() | Add a restriction in HAVING clause (each restriction will be separated by AND statement) |
| innerJoin() | Add INNER JOIN clause, E.g. <code>\$this->innerJoin('product p ON ...')</code> |
| join() | Add JOIN clause, E.g. <code>\$this->join('RIGHT JOIN '._DB_PREFIX_.'product p ON ...')</code> ; |
| leftJoin() | Add LEFT JOIN clause |
| limit() | Limit results in query |
| orderBy() | Add an ORDER BY restriction |
| select() | Add fields in query selection |
| where() | Add a restriction in WHERE clause (each restriction will be separated by AND statement) |
| ... | ... |

Doctrine

Le mapping

Doctrine intègre un système de mapping pour associer les propriétés d'une classe avec les colonnes d'une table dans la base de données.



Les entités

Pour utiliser le mapping, vous devez ajouter des "métadonnées" permettant de décrire l'association entre une propriété et une colonne.

Exemple avec l'Entity *Attribute* :

```
<?php

/**
 * Attribute
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="PrestaShopBundle\Entity\AttributeRepository")
 */
class Attribute
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id_attribute", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
}
```

Les getters et les setters peuvent être automatiquement générés par *Doctrine* avec la commande :

```
php app/console doctrine:generate:entities PrestaShopBundle/Entity/Attribute
```

PrestaShop et Doctrine

Les dépendances de PrestaShop avec Doctrine :

- "doctrine/dbal": "~2.5.3",
- "doctrine/common": "~2.5.3",
- "doctrine/orm": "~2.5.3",
- "doctrine/doctrine-bundle": "1.5.2",

Le service EntityManager

Le service *EntityManager* se charge d'exécuter les requêtes SQL.

Il assure également la persistance des objets dans la base.

Pour accéder à l'EntityManager il suffit d'appeler la fonction *getManager()* ou d'utiliser le container de services.

Exemple :

```
<?php

// get Entity Manager in Controller
$em = $this->getDoctrine()->getManager();
// get Entity Manager from services
$em = $this->get('doctrine.orm.entity_manager');
```

Les Repository

A partir de l'EntityManager, vous pouvez récupérer une entité par l'intermédiaire d'un *Repository*

Il existe un *Repository* par type d' *Entity*

Exemple d'utilisation d'un *Repository* dans

/src/PrestaShopBundle/Translation/Loader/DatabaseTranslationLoader.php :

```
<?php

$LANGRepository = $em->getRepository('PrestaShopBundle:Lang');
```

Créer un enregistrement

Insérer une nouvelle valeur.

Exemple avec l'utilisation de l'entité *ModuleHistory* :

```
<?php

$moduleHistory = new ModuleHistory();

$moduleHistory->setIdEmployee($currentEmployeeID);
$moduleHistory->setIdModule($moduleAccessedID);
$moduleHistory->setDateUpd(new \DateTime(date('Y-m-d H:i:s')));

$em = $this->getDoctrine()->getManager();
$em->persist($moduleHistory);
$em->flush();
```

Modifier un enregistrement

Exemple avec l'Entity *Lang* :

```
<?php

$em = $this->getDoctrine()->getManager();

$lang = $em->getRepository('PrestaShopBundle:Lang')->find(1);
$lang->setActive(false);
$lang->flush();
```

Utiliser des transactions

L'EntityManager propose d'isoler les modifications par transaction.

Vous pouvez enregistrer ou annuler toutes les modifications sur plusieurs *Entity*

```
<?php

$conn->beginTransaction();

try {
    // do stuff
    $conn->commit();
} catch (\Exception $e) {
    $conn->rollBack();
    throw $e;
}
```

Lire un enregistrement

A partir d'un *Repository* vous disposez de nombreuses fonctions pour lire un ou des enregistrements.

EntityRepository supporte les méthodes suivantes :

- *findOne(\$id)* : finds an entity by its primary key
- *findOneByXYZ(\$XYZFieldValue)* : finds zero or one entity
- *findByXYZ(\$XYZFieldValue)* : finds zero or more entities
- *findOneBy(array \$conditions)* : finds zero or one entity that matches all of the passed conditions
- *findBy(array \$conditions)* : finds zero or more entities that match all of the passed conditions

Exemple d'utilisation :

```
<?php

// With repository as a service
$langRepository = $this->get('prestashop.core.admin.lang.repository');
$locale = $langRepository->findOneBy([
    'iso code' => 'en',
    'active' => true
])->getLocale();

// With entity manager
$entityManager = $this->get('doctrine.orm.entity manager');
$langRepository = $entityManager->getRepository('Lang');
$locale = $langRepository->findOneBy([
    'iso_code' => 'en',
    'active' => true
])->getLocale();
```

Impact sur le développement

Coexistence de plusieurs classes

Par exemple pour accéder à la table Lang :

- Lang (*ObjectModel*) dans /classes/Language.php
- Lang (*Doctrine*) dans PrestaShopBundle/Entity/Lang.php

Problème sur les surcharges.

Surveiller :

- La migration progressive des modèles avec Doctrine.
- La création de nouveaux modèles.

Privilégier l'utilisation de Doctrine pour les nouveaux développements.

Web services

Principes

REST

Representational state transfer (REST) n'est pas un protocole, un standard ou encore un format.

Il s'agit d'un style d'architecture pour les systèmes distribués.

Un des concepts importants de REST est la notion de ressource :

- Chaque ressource est accessible par une URL.
- Chaque ressource peut contenir un lien vers une ou plusieurs autres ressources.

Les ressources sont accessibles via un ensemble uniforme de commandes fournies par *HTTP* (essentiellement GET, POST, PUT et DELETE) qui permettent de spécifier l'opération à effectuer sur une ressource.

Activer les Web Services

Pour utiliser les Web Services vous devez les activer et créer les accès dans le back-office de PrestaShop. Rendez-vous dans le menu **Paramètres avancés > Service Web** .

Pour vérifier la bonne activation, rendez-vous sur la page <http://mystore.com/api/> , et saisissez la clé dans la zone utilisateur (aucun mot de passe).

Remarque : Afin que la clé ne puisse être devinée, utiliser le bouton "Générer". Si vous définissez vous même la clé, s'assurer qu'elle soit suffisamment sécurisée et que ses droits sont limités.

Utilisation

Toutes les ressources sont accessibles par une url.

Exemples :

- <http://mystore.com/api/adresses> -> donne la liste des adresses
- <http://mystore.com/api/adresses/1> -> donne la première adresse

Utiliser les web services

Installer l'extension *CURL* pour PHP (extension=php_curl.dll).

Utiliser la librairie **PSWebServiceLibrary.php**

Disponible sur GitHub à l'adresse <https://github.com/PrestaShop/PrestaShop-webservice-lib>

Accéder aux web services

Créer une instance de *PrestaShopWebservice* qui prend dans son constructeur 3 paramètres :

- Chemin racine de la boutique (ex. : <http://mystore.com/>)
- La clé d'authentification (ex. : ZR92FNY5UFRERNI3O9Z5QDHWKTP3YIIT)
- Un booléen indiquant si le service web doit utiliser son mode debug.

Une fois l'instance créée vous pouvez accéder aux méthodes :

- *get()* : pour lire une ressource
- *add()* : pour envoyer une ressource
- *edit()* : pour modifier une ressource
- *delete()* : pour supprimer une ressource

Lire des données

Pour lire une ressource, utiliser la fonction *get()* en précisant dans le tableau de paramètre :

- resource : le type de ressource
- id : l'id éventuelle de la ressource

La fonction *get* retourne un objet SimpleXML contenant l'ensemble des informations demandées.

Exemple PHP :

```
<?php
$opt['resource'] = 'customers';
$xml = $webService->get($opt);
```

Résultat XML :

```
<?xml>
<prestashop>
  <customers>
    <customer>
      ID du client
    </customer>
    ...Autres balises client
  </customers>
</prestashop>
```

La fonction *children()* permet de récupérer tous les enfants d'une balise.

```
<?php
$resources = $xml->customers->children();
foreach ($resources as $key => $resource) {
    echo 'Nom du champ : '.$key.' - Valeur : '.$resource.'<br>';
}
```

Modifier des données

Pour modifier une ressource, utiliser la fonction *edit()* en précisant dans le tableau de paramètres :

- resource : le type de ressource
- id : l'id de la ressource
- putXml : l'objet xml avec les données modifiées

Exemple :

```
<?php
$opt = [
    'resource' => 'customers',
    'id' => 1,
    'putXml' => $myresource->asXML(),
];
$xml = $webService->edit($opt);
```

Supprimer des données

Pour supprimer une ressource, utiliser la fonction *delete()* en précisant dans le tableau de paramètres :

- resource : le type de ressource
- id : l'id de la ressource

Exemple :

```
<?php
try {
    $ws = new PrestaShopWebservice(
        'http://mystore.com/',
        'ZQ88PRJX5VWQHCWE4EE7SQ7HPNX00RAJ',
        false
    );
    $xml = $ws->delete(
        [
            'resource' => 'orders',
            'id' => 1
        ]
    );
    echo 'Successfully deleted.';
} catch (PrestaShopWebserviceException $ex) {
    echo 'Error : '.$ex->getMessage();
}
```

Ajouter des données

Pour ajouter une ressource, utiliser la fonction `add()` en précisant dans le tableau de paramètres :

- resource : le type de ressource
- putXml : l'objet xml avec les données à insérer

Exemple :

```
<?php
$opt = [
    'resource' => 'customers',
    'putXml' => $myressource->asXML(),
];
$xml = $webService->add($opt);
```

Il est possible d'initialiser un objet simpleXML avec une ressource vide.

Exemple :

```
<?php
$xml = $webService->get(
    [
        'url' => 'http://mystore.com/api/customers?schema=blank'
    ]
);
```

Les données

Les images

L'accès aux images se fait via l'entité "images".

Plusieurs types d'images existent :

- Générale à la boutique **/api/images/general**
- Produit **/api/images/products**
- Catégorie **/api/images/categories**
- Fabricant **/api/images/manufacturers**
- Fournisseur **/api/images/suppliers**
- Magasin **/api/images/stores**

Exemple :

```
<?php
$opt['url'] = 'http://mystore.com/api/images/products/5/10';
$xml = $webService->get($opt);
```

Les prix

Il est possible de récupérer les prix personnalisés d'un produit en utilisant des paramètres.

Exemple : Je veux le prix du produit 1 hors taxe dans un champs webservice ayant pour nom "mon_prix" : **/api/products/1?price[mon_prix][use_tax]=0**

Voici la liste des paramètres permettant de récupérer un prix personnalisé :

- country
- state
- currency
- group
- quantity
- decimals
- product_attribute
- use_tax
- use_reduction
- use_ecotax
- only_reduction

Utilisation avancée

Filtrer et trier les informations

Exemples :

- N'inclure que les noms et prénoms des clients "customers" ayant l'id 1 et 5 :
/api/customers/?display=[firstname,lastname]&filter[id]=[1|5]
- N'inclure que la date de naissance du client ayant pour nom "John" et prénom "DOE" :
/api/customers/?display=[birthday]&filter[firstname]=[John]&filter[lastname]=[DOE]
- N'inclure que les noms des constructeurs "manufacturers" dont le nom commence par "Appl" :
/api/manufacturers/?display=[name]&filter[name]=[appl]%
- Trier les clients "customers" en ordre alphabétique du nom et n'inclure que les 5 éléments à partir du 10ème : **/api/customers?display=full&sort=[lastname_ASC]&limit=9,5**

Limiter le nombre de zones à lire

Ne lire que les champs "name" et "value" de la ressource "configurations"

/api/configurations/?display=[name,value]

Exemple PHP :

```
<?php
$opt = [
    'resource' => 'configurations',
    'display' => '[name,value]'
];
```

Web service et ObjectModel

Gérer des liens entre différentes ressources

Il est possible de gérer des liens entre différentes ressources en utilisant le tableau *\$webserviceParameters*

```
<?php
$webserviceParameters = array(
    'fields' => array(
        'id_currency' => array(
            'xlink_resource' => 'currencies'
        ),
        'id_customer' => array(
            'xlink_resource' => 'customers'
        )
    )
);
```

Dans cet exemple, une balise xlink vers la devise et le client seront automatiquement ajoutés.

```
<id_currency xlink:href="http://mystore.com/api/currencies/1">
    <![CDATA[ 1 ]] >
</id_currency>
<id_customer xlink:href="http://mystore.com/api/customers/3">
    <![CDATA[ 3 ]] >
</id_customer>
```

Gérer des liens entre différentes ressources

Utiliser *\$webserviceParameters['associations']* pour ajouter automatiquement des ressources associées.

```
<?php
$webserviceParameters['associations'] = array(
    'cart_rows' => array(
        'resource' => 'cart_row',
        'virtual_entity' => true,
        'fields' => array(
            'id_product' => array(
                'required' => true,
                'xlink_resource' => 'products'
            ),
            'id_product_attribute' => array(
                'required' => true,
                'xlink_resource' => 'combinations'
            ),
            'quantity' => array('required' => true)
        )
    )
);
```

Exemple de résultat

```
<associations>
  <cart_rows virtual_entity="true" node_type="cart_row">
    <cart_row>
      <id_product xlink:href="http://mystore.com/api/products/4">
        <![CDATA[ 4 ]]>
      </id_product>
      <id_product_attribute xlink:href="http://mystore.com/api/combinations/0"
    >
      <![CDATA[ 0 ]]>
      </id_product_attribute>
      <quantity>
        <![CDATA[ 1 ]]>
      </quantity>
    </cart_row>
  </cart_rows>
</associations>
```

Définir le getter et le setter d'une ressource

Il est possible d'associer un *getter* et un *setter* ou d'interdire la lecture et la mise à jour d'une zone accessible par Web service.

Exemple :

```
<?php
$webserviceParameters = array(
    'fields' => array(
        'id tax rules group' => array(
            'getter' => 'getIdTaxRulesGroup',
            'setter' => 'setTaxRulesGroup'
        ),
        'quantity' => array(
            'getter' => false,
            'setter' => false
        ),
    ),
);
```

Gérer l'ajout et la modification de ressources

Utiliser `$webserviceParameters['objectMethods']` pour définir les fonctions utilisées lors de l'ajout ou la modification de ressources.

Exemple :

```
<?php
$webserviceParameters['objectMethods'] = array(
    'add' => 'addWs',
    'update' => 'updateWs'
);
```

Templating

Templating dans PrestaShop

Moteurs de templates

PrestaShop utilise 2 moteurs de template :

- *Smarty* pour le front-office et le back-office legacy.
- *Twig* pour le nouveau back-office 1.7 (Symfony)

Remarque : Depuis la version 1.7.6, PrestaShop utilise *Twig* pour la génération des templates d'emails

Smarty

Présentation

Smarty



Smarty est un moteur de rendu :

- Il facilite la séparation entre la logique applicative et la présentation.
- Propose une syntaxe propre et condensée dédiée au design.
- Fonctionnement par compilation puis cache.

Liens connexes

Smarty (<https://www.smarty.net/>)

Source de Smarty (<https://github.com/smarty-php/smarty/>)

Installation de Smarty

Smarty est installé automatiquement avec PrestaShop.

Vous pouvez installer manuellement Smarty avec Composer.

Pour installer Smarty manuellement vous pouvez utiliser la commande suivante :

```
composer require smarty/smarty 3.1.31
```

Cette installation va créer un dossier **smarty** dans le répertoire **/vendor**

Remarque : Vous devez avoir préalablement installé Composer.

Utilisation de templates

Smarty utilise des templates.

Dans un template, le contenu qui est situé en dehors des délimiteurs est affiché comme contenu statique. Il est inchangé.

Lorsque Smarty rencontre une instruction entre deux balises "{" et "}", il tente de la comprendre et affiche le résultat à la place de l'instruction.

Exemple de template :

```
<p>No {doSomething} has ever made a mistake or distorted information.</p>
```

Si la fonction doSomething affiche "9000 COMPUTER", nous avons le résultat suivant :

```
<p>No 9000 COMPUTER has ever made a mistake or distorted information.</p>
```

Utilisation en PHP

Un template Smarty est utilisé à partir de code PHP.

Exemple de code :

```
$smarty->assign('name', '9000 COMPUTER');  
echo $smarty->display('mistake.tpl');
```

Si nous avons le template **mistake.tpl** suivant :

```
<p>No {$name} has ever made a mistake or distorted information.</p>
```

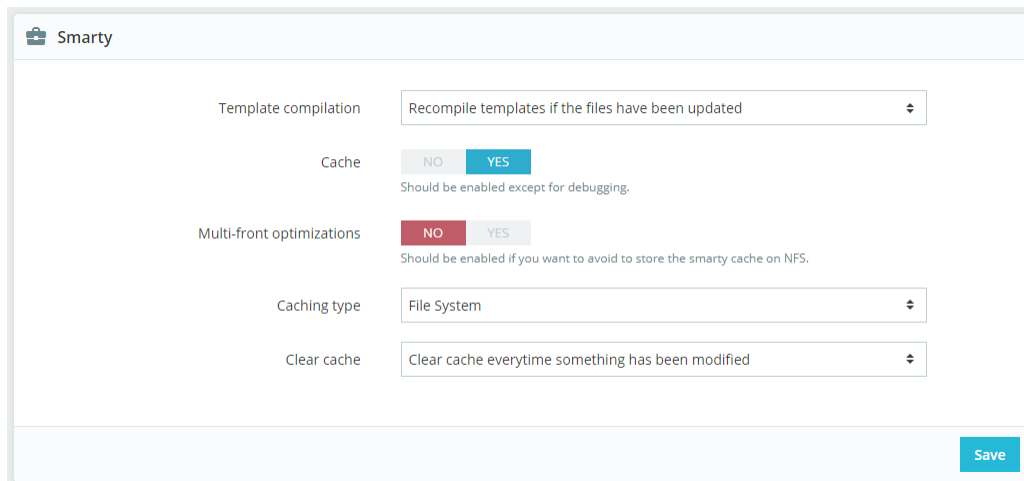
Nous avons le résultat suivant :

```
<p>No 9000 COMPUTER has ever made a mistake or distorted information.</p>
```

Paramétrage de Smarty

PrestaShop propose un écran de paramétrage pour Smarty dans le back-office.

Rendez-vous dans le menu **Paramètres avancés > Performances**



The screenshot shows the 'Smarty' configuration page in PrestaShop. It includes the following settings:

- Template compilation:** A dropdown menu set to 'Recompile templates if the files have been updated'.
- Cache:** Radio buttons for 'NO' and 'YES' (selected). Below it, a note says 'Should be enabled except for debugging.'
- Multi-front optimizations:** Radio buttons for 'NO' (selected) and 'YES'. Below it, a note says 'Should be enabled if you want to avoid to store the smarty cache on NFS.'
- Caching type:** A dropdown menu set to 'File System'.
- Clear cache:** A dropdown menu set to 'Clear cache everytime something has been modified'.
- A **Save** button is located at the bottom right of the configuration area.

Configuration

PrestaShop utilise le fichier **/config/smarty.config.php** pour configurer Smarty.

Configuration par défaut :

- Répertoire de stockage pour les templates compilés : **/var/cache/(env)/smarty/compile**
- Répertoire de stockage pour les templates en cache : **/var/cache/(env)/smarty/cache**

Fonctions et filtres initialisés : escape, truncate, l, hook, json_encode, json_decode, dateFormat, boolval, cleanHtml, classname, classnames et url.

Remarque : Les fonctions et filtres initialisés dans ce fichier sont disponibles sur le Front et le back-office de PrestaShop.

Configuration spécifique pour le front-office

Le front-office de PrestaShop utilise le fichier `/config/smartyfront.config.php` pour configurer Smarty.

Configuration par défaut :

- Répertoire des templates : `/themes/{theme_actif}/templates`
- Répertoire des plugins : `/themes/{theme_actif}/plugins`

Fonctions et filtres initialisés : widget, render, form_field et widget_block.

Remarque : Les fonctions et filtres initialisés dans ce fichier sont disponibles sur le front-office de PrestaShop.

Configuration spécifique pour le back-office

Le back-office de PrestaShop utilise le fichier `/config/smartyadmin.config.php` pour configurer Smarty.

Configuration par défaut :

- Force compilation des templates s'ils ont été modifiés.

Fonctions et filtres initialisés : toolsConvertPrice, convertPrice, convertPriceWithCurrency, displayWtPrice, displayWtPriceWithCurrency, displayPrice, convertAndFormatPrice, getAdminToken, displayAddressDetail, getWidthSize, getHeightSize, addJsDef, addJsDefL et secureReferrer.

Remarque : Les fonctions et filtres initialisés dans ce fichier sont disponibles sur le back-office de PrestaShop.

La compilation des templates

La compilation des templates Smarty permet de transformer un template (fichier .tpl) en programme php.

Avantage : Smarty n'a pas besoin de relire un template pour l'exécuter.

La mise en cache des templates

Le cache Smarty permet d'accélérer l'affichage d'un template en sauvegardant le dernier résultat calculé dans un fichier. Le dernier résultat calculé est toujours réaffiché. Pour recalculer l'affichage, le cache doit être vidé.

Avantage : Le code PHP nécessaire au calcul du template n'est pas exécuté à chaque affichage de la page.

Exemple : Le module ps_mainmenu utilise le cache de Smarty.

Remarque : Le cache Smarty est désactivé sur le back-office de PrestaShop. Pour des questions de performance, le cache Smarty doit toujours être activé sur le front-office.

Bases syntaxiques

Délimiteurs

Smarty utilise des délimiteurs pour « reconnaître » les instructions à exécuter. Dans Smarty, les délimiteurs sont les accolades { et } (ils peuvent être modifiés).

Smarty reconnaît les instructions suivantes :

- {*\$var_name*} pour afficher le contenu de la variable *var_name*
- {** comment **} pour écrire un commentaire.
- {*function_name param_name=value*} pour exécuter une fonction avec un paramètre.
- {*function*}...{/*function*} pour exécuter une fonction sur le contenu d'un bloc.

Les commentaires

Les commentaires utilisent les délimiteurs plus une étoile.

```
{* Commentaire sur une ligne *}

{*
Commentaire sur plusieurs
lignes
*}

{* Commentaire de {$code} Smarty *}
```

Remarque : Les commentaires Smarty ne sont PAS affichés dans le résultat du template. Il sont donc différent des <!-- commentaires --> en HTML.

Désactiver smarty

Les balises {literal} permettent à un bloc de données d'être pris tel quel, sans interprétation par Smarty.

```
{literal}
  <script type="text/javascript">
    function maFonction()
    {
    }
  </script>
{/literal}
```

Remarque : Cette instruction permet d'insérer des "{" ou "}" dans un template sans générer d'erreur d'interprétation de Smarty.

Les variables

Smarty permet d'utiliser des variables pour insérer du contenu dynamique dans une page. Le nom d'une variable peut contenir des nombres, des lettres et des underscores. Une variable commence toujours par dollar.

Exemple de variable dans un template :

```
{assign name="mistake" value="has ever made a mistake or distorted information"}
{$computer="9000"}

<p>No {$computer} COMPUTER {$mistake}.</p>
```

Résultat :

```
<p>No 9000 COMPUTER has ever made a mistake or distorted information.</p>
```

Les conditions

Smarty possède un système conditionnel if / else / elseif.

```
{if $coffee eq 'good'}
    {* Stagiaires heureux *}
{elseif $coffee == 'very good'}
    {* Stagiaires super heureux *}
{else}
    {* Stagiaires malheureux *}
{/if}
```

Les boucles

Exemple de code PHP :

```
$product_list = array(
    23 => array('no' => 2456, 'label' => 'Salad'),
    96 => array('no' => 4889, 'label' => 'Cream')
);

$smarty->assign('products', $product_list);
```

Utilisation dans Smarty

```
<ul>
    {foreach from=$products item=product}
        <li><p><a href="product.php?id={$product.no}">{$product.label}</p></li>
    {/foreach}
</ul>
```

Il est possible de connaître l'indice de l'enregistrement dans une boucle avec *\$smarty.foreach* suivi du nom de la boucle.

Exemple :

```
{foreach from=$products key=id_product item=product name=listproduct}
    {if $smarty.foreach.listproduct.first}Premier{/if}
    {if $smarty.foreach.listproduct.last}Dernier{/if}
    Index : {$smarty.foreach.listproduct.index}
    {if $smarty.foreach.listproduct.index % 2}Ligne paire{/if}
{/foreach}
```

Les captures

Capture permet de capturer la sortie d'un élément sans l'afficher :

```
{capture name='maCapture'} ... {/capture}
```

Pour utiliser le contenu, utilisez la variable `$smarty.capture.<nom>`. Vous pouvez tester une capture : `{if $smarty.capture.<nom> != ""}` ou `{if isset($smarty.capture.<nom>)}`. Capture permet également d'auto-assigner une variable :

```
{capture name='maCapture' assign='maVar'}  
...  
{/capture}  
{* Pour utiliser le contenu, utiliser la variable : $maVar *}
```

L'inclusion

L'inclusion permet d'inclure un « template » dans un autre.

Exemple d'inclusion

```
{* Exemple *}  
{include file='steps.tpl'}  
  
{* Exemple avec contenu dans une variable *}  
{include file='text.tpl' assign='MyText'}  
{$MyText}
```

L'héritage

L'héritage permet de réutiliser du HTML dans différentes pages.

Exemple de fichier 'parent' qui contient la structure de la page :

```
{* parent.tpl *}  
<html>  
<head>  
<body>  
  {block name="bodypart"}Default Text{/block}  
</body>  
</html>
```

Exemple de fichier 'enfant' qui redéfinit une partie de la page :

```
{* child.tpl *}  
{extends file="parent.tpl"}  
  
{block name="bodypart"}  
  {$smarty.block.parent}  
  Add this  
{/block}
```

Les variables dans Smarty

Utiliser les variables

Vous pouvez utiliser des variables pour insérer du contenu dynamique dans la page. Le nom des variables est précédé d'un dollar.

Exemple pour afficher la valeur de la variable `var_name` :

```
{$var_name}
```

Echappement

A partir de PrestaShop 1.7, toutes les variables sont échappées par défaut. Pour ne pas échapper une variable vous devez utiliser *nofilter*

Exemple :

```
<section id="content" class="page-content page-cms page-cms-{$cms.id}">
    {$cms.content nofilter}
</section>
```

Les filtres

On peut appliquer des « filtres de sortie » sur une variable.

Exemples :

```
{* Exemple pour mettre en minuscule *}
{$foo|lower}

{* Exemple avec plusieurs filtres *}
{$foo|lower|capitalize|truncate:10:'...'}
```

Créer une variable

Vous pouvez créer des variables directement dans un template.

Exemples :

```
{assign var="formation" value="Smarty"}
{* La variable {$formation} affichera : Smarty *}

{$formation="Smarty"}
{* idem en version courte *}
```

La variable \$smarty

\$smarty est une « variable interne » qui permet de récupérer des informations :

- Utilisée pour « capture »
- Utilisée pour « foreach »
- Utilisée pour les données « GET » : `{$smarty.get.<nom>}`
- Utilisée pour les données « POST » : `{$smarty.post.<nom>}`
- Utilisée pour afficher le « timestamp » courant : `{$smarty.now}` ou formaté `{$smarty.now|date_format:'%d-%m-%Y %H:%M:%S'}`
- Utilisée pour les constantes PHP : `{$smarty.const.<nom de la constante>}`

Exemples d'utilisation pour les constantes PHP :

```
{$smarty.const._DB_SERVER_} / {$smarty.const._DB_USER_}
```

La variable \$link

\$link permet d'accéder à la classe *Link* de PrestaShop et de créer une URL en fonction du contexte :

- URL vers un produit (`getProductLink`), une catégorie (`getCategoryLink`), une page CMS

- (getCMSLink), etc.
- URL vers une image produit (getImageLink), le logo d'un fournisseur (getSupplierImageLink), etc.
- URL vers une page de la boutique (getPageLink)

Exemples d'utilisation :

```
<a href="{ $link->getProductLink($product.id) }">{$product.name}</a>
```

La fonction `$link->getAdminLink()`

La fonction `getAdminLink` peut prendre en charge de la migration progressive des contrôleurs back-office sous Symfony.

Attention, pour passer des paramètres, vous ne devez pas écrire :

```
<a href="
    {$link->getAdminLink('AdminOrders')}
    &id_order={$order->id|intval}
    &vieworder' }
">View</a>
```

Vous devez utiliser les paramètres pour passer des paramètres à l'url.

Exemple :

```
<a href="
{$link->getAdminLink(
    'AdminOrders',
    true,
    [],
    [
        'id_order' => $order->id|intval,
        'vieworder' => 1
    ]
)}">View</a>
```

Les filtres

Utiliser les filtres

Les filtres peuvent être appliqués aux variables, aux paramètres des fonctions ou aux chaînes de caractères.

Pour appliquer un filtre, vous devez utiliser `|` (pipe) suivi du nom du filtre.

Exemple :

```
{* var filter *}
{$title|upper}

{* var filter with parameters *}
{$title|truncate:40:'...'}

{* function parameter with filter *}
{myfunction param={$title|upper}}
```

Les filtres smarty

Les filtres sont chargés automatiquement depuis le répertoire plugins de Smarty
/vendor/smarty/libs/plugins

Ils peuvent être ajoutés explicitement avec la fonction *smartyRegisterFunction()* .

- Consultez le fichier **/config/smarty.config.php** pour découvrir les filtres ajoutés par PrestaShop.
- Les filtres sont chargés à la demande.

Remarque : Vous pouvez également utiliser des plugins Smarty en les ajoutant dans le répertoire plugins du thème actif de votre boutique.

Les filtres PHP

Toutes les fonction PHP peuvent être utilisées comme filtres, sans autre déclaration.

Attention, l'ordre des paramètres de la fonction n'est pas toujours celui attendu.

Exemple :

```
{* PHP print_r *}
{$myArray|print_r}

{* PHP str_repeat *}
{'='|str_repeat:80}
```

Remarque : Il est possible d'activer des sécurités dans Smarty pour limiter les fonctions disponibles.

lower

Permet d'afficher le contenu d'une variable en minuscules.

Exemple :

```
{ $title="No 9000 COMPUTER has ever made a mistake or distorted information." }
<p>{$title|lower}</p>
```

Résultat :

```
<p>no 9000 computer has ever made a mistake or distorted information.</p>
```

replace

Permet de remplacer un texte par un autre.

Exemple :

```
{ $title='No 9000 COMPUTER has ever made a error or distorted information.' }
<p>{$title|replace:'error':'mistake'}</p>
```

Résultat :

```
<p>no 9000 computer has ever made a mistake or distorted information.</p>
```

capitalize

Met la première lettre de chaque mot d'une variable en majuscule.

C'est l'équivalent de la fonction PHP `ucfirst()`

Exemple :

```
{ $title="No 9000 COMPUTER has ever made a mistake or distorted information." }  
<p>{ $title|capitalize }</p>
```

Résultat :

```
<p>No 9000 Computer Has Ever Made A Mistake Or Distorted Information.</p>
```

escape

Ce filtre va échapper et décoder une chaîne de caractère.

Exemple de passage de variable en PHP :

```
$smarty->assign(array(  
    'myContent' => 'No 9000 \'COMPUTER\' has ever made a mistake or distorted  
    information.',  
));  
$smarty->fetch('module:module_name/views/templates/hook/hello.tpl')
```

Exemple de template :

```
<p>{ $myContent nofilter }</p>  
<p>{ $myContent|escape:'html' }</p>
```

Résultat :

```
<p>No 9000 'COMPUTER' has ever made a mistake or distorted information.</p>  
<p>No 9000 &#039;COMPUTER&#039; has ever made a mistake or distorted informat  
ion.</p>
```

truncate

Permet de couper une chaîne de caractère à une certaine longueur.

Un paramètre optionnel permet d'ajouter automatiquement des caractères à la fin de la chaîne coupée.

Les caractères ajoutés sont pris en compte dans le calcul de la longueur de la chaîne.

Exemple :

```
{ *           1           2           3           4           5           6  
{ *      123456789012345678901234567890123456789012345678901234567890123456 *  
}  
{ $title="No 9000 COMPUTER has ever made a mistake or distorted information." }  
<p>{ $title|truncate:16 }</p>  
{ $title="No 9000 COMPUTER has ever made a mistake or distorted information." }  
<p>{ $title|truncate:43:"..." }</p>
```

Résultat :

```
<p>No 9000 COMPUTER</p>
<p>No 9000 COMPUTER has ever made a mistake...</p>
```

json_encode

Convertit un tableau en une chaîne encodée au format JSON.

Exemple de passage de variable en PHP :

```
$smarty->assign(array(
    'arrData' => array(
        'a' => 1,
        'b' => 2,
        'c' => 3
    )
));
$smarty->fetch('module:module_name/views/templates/hook/hello.tpl')
```

Exemple de template :

```
{assign var="jsonData" value=$arrData|json_encode}
<p>{$jsonData}</p>
```

Résultat :

```
<p>{"a":1,"b":2,"c":3}</p>
```

json_decode

Décode une chaîne encodée au format JSON pour convertir en tableau.

Exemple de passage de variable en PHP :

```
$smarty->assign(array(
    'jsonData' => '{"a":1,"b":2,"c":3}'
));
$smarty->fetch('module:module_name/views/templates/hook/hello.tpl')
```

Exemple de template :

```
{assign var="arrData" value=$jsonData|json_decode}
<ul>
{{foreach from=arrData item=oneData key=recKey name=recName}
    <li><p>{$recKey} -> $recName</p></li>
{/foreach}}
</ul>
```

Résultat :

```
<ul>
    <li><p>a -> 1</p></li>
    <li><p>b -> 2</p></li>
    <li><p>c -> 3</p></li>
</ul>
```

cleanHtml

Ce filtre va garantir que votre chaîne ne contient pas de JavaScript.

Il permet de se protéger contre les attaques de type Cross-Site Scripting (XSS).

Exemple de passage de variable en PHP :

```
$smarty->assign(array(
    'cleanContent' => 'No 9000 COMPUTER has ever made a mistake or distorted
information.',
    'badContent1' => '<script>document.getElementById("badContent1").innerHTM
L="badContent 1 XSS attack."</script>',
    'badContent2' => '<script>document.getElementById("badContent2").innerHTM
L="badContent 2 XSS attack."</script>',
));
$smarty->fetch('module:module_name/views/templates/hook/hello.tpl')
```

Exemple de template :

```
<p>{$cleanContent nofilter}</p>
<p>{$cleanContent|cleanHtml nofilter}</p>
<p id="badContent1">The 9000 series is the most reliable computer ever made.<
/p>
<p id="badContent2">The 9000 series has a perfect operational record.</p>
{$badContent1 nofilter}
{$badContent2|cleanHtml nofilter}
```

Résultat :

```
<p>No 9000 COMPUTER has ever made a mistake or distorted information.</p>
<p>No 9000 COMPUTER has ever made a mistake or distorted information.</p>
<p id="badContent1">badContent 1 XSS attack.</p>
<p id="badContent2">The 9000 series has a perfect operational record.</p>
```

classname

Ce filtre va garantir que votre chaîne est un nom de classe valide :

- La chaîne sera en minuscule.
- Tous les caractères non-ASCII (tels que les caractères accentués) seront remplacés par leur équivalent ASCII.
- Tous les caractères non alphanumériques seront remplacés par un seul tiret.
- Un seul tiret consécutif sera utilisé.

Exemple :

```
{assign var=attr value='Here-Is_a-Classname---'}
<div class="{ $attr|classname}">
    No 9000 COMPUTER has ever made a mistake or distorted information.
</div>
```

Résultat :

```
<div class="here-is-a-classname">
    No 9000 COMPUTER has ever made a mistake or distorted information.
</div>
```

classnames

Ce filtre utilise un tableau de classe avec une valeur vrai ou faux pour sélectionner celles qui doivent être utilisées.

Exemple :

```
{assign var=attrs value= [
    "lang-fr" => true,
    "rtl" => false,
    "country-FR" => true,
    "currency-EUR" => true,
    "layout-full-width" => true,
    "page-index" => true
]}
<body class="{\$attrs|classnames}">
```

Résultat :

```
<body class="lang-fr country-fr currency-eur layout-full-width page-index">
```

convertAndFormatPrice

Ce filtre permet de convertir et de formater un montant dans une devise.

Equivalent à :

```
{displayPrice price=Tools::ps_round(Tools::convertPrice(\$customerStats['total_orders'], \$currency), 2) currency=\$currency->id}
```

Exemple d'utilisation :

```
{\$cotax_tax_exc|convertAndFormatPrice}
```

secureReferrer

Permet de sécuriser une url en vérifiant qu'elle provient bien de la boutique.

Exemple :

```
{\$smarty.server.HTTP_REFERER|escape:'htmlall':'UTF-8'|secureReferrer}
```

Les fonctions

Utiliser les fonctions

Il existe deux types de fonctions :

- Les fonctions natives, qui sont relatives au traitement interne de Smarty (if, include, foreach, etc).
- Les fonctions additionnelles, qui implémentées par l'intermédiaire de plugins ou depuis le code PHP (url, widget, hook, etc.)

Les fonctions natives et les fonctions additionnelles utilisent la même syntaxe, dans les templates.

```
{fonction_name param_1=value param_2=value}
```

Les fonctions natives

Smarty est fourni en standard avec plusieurs fonctions natives.

Exemple de fonction native : capture, foreach, if, include, literal, ...

Les fonctions natives sont chargées depuis le répertoire plugins de smarty
/vendor/smarty/libs/plugins

Les fonctions additionnelles

PrestaShop ajoute de nombreuses fonctions à Smarty .

Exemple de fonction additionnelles : url, render, hook, widget, ...

Les fonctions additionnelles sont chargées dans les fichiers de configuration pour Smarty
/config/smarty.config.inc.php

Vous pouvez ajouter des fonctions à Smarty depuis un module avec la fonction
smartyRegisterFunction

Remarque : Vous pouvez également utiliser des plugins Smarty en les ajoutant dans le répertoire **/themes/{my_theme}/plugins** du thème actif de votre boutique.

Les fonctions PHP

Toutes les fonction PHP peuvent être utilisées comme fonctions, sans autre déclaration.

Exemple :

```
{if !is_dir($moduleDirectory) }  
    {assign var=fp value=fopen($filename,"w")}  
    {assign var=fw value=fwrite($fp, $content)}  
    {assign var=fc value=fclose($fp)}  
{/if}
```

Remarque : Il est possible de limiter les fonctions disponibles dans Smarty.

Les paramètres

Les paramètres des fonctions Smarty sont très proches des attributs des balises HTML.

Les valeurs numériques n'ont pas besoin de guillemets.

Les guillemets sont recommandés pour les chaînes de caractères.

- Pour les booléens vous pouvez utiliser true, on, ou yes, et false, off, ou no.
- Smarty peut reconnaître des variables dans une chaînes de caractères entre guillemets.
- Les filtres doivent toujours être appliquer à l'extérieur des guillemets.

Exemple :

```
<p>{fonction_display param_1="No $computer"}</p>  
<p>{fonction_display param_1="No $computer"|lower}</p>
```

Résultat avec "computer" dans la variable \$computer :

```
<p>No COMPUTER</p>  
<p>no computer</p>
```

La fonction url

La fonction *url* permet de générer une url (En remplacement de \$link depuis PrestaShop 1.7.0.0).

Exemple d'utilisation :

```
{url entity=cms id=2}
{url entity=category id=3}
{url entity=address id=1 params=['delete' => true]}
{url entity='sf' route='admin_module_manage'}
```

Résultats :

- <http://prestashop.ps/en/content/2-legal-notice>
- <http://prestashop.ps/en/3-women>
- http://prestashop.ps/en/address?id_address=1&delete=1
- <http://prestashop.ps/en/admin/module/manage>

La fonction `render`

La fonction `render` permet d'utiliser une instance spécifique de Smarty.

Exemple pour le formulaire de saisie d'une adresse :

```
<div class="address-form">
  {render template="customer/_partials/address-form.tpl" ui=$address_form}
</div>
```

La fonction `form_field`

La fonction `form_field` permet d'ajouter simplement une zone de saisie dans un formulaire.

Exemple de création d'une zone :

```
$field = array(
    'type' => 'hidden',
    'name' => 'my-hidden-field',
    'value' => 'my-value',
);
$smarty->assign('field', $field);
```

Pour créer la zone :

```
{form_field field=$field}
```

La fonction `form_field` utilise le template `/templates/_partials/form-fields.tpl` pour créer les zones de saisie.

La fonction `hook`

La fonction `hook` permet de créer un hook dans un template.

Exemple d'utilisation :

```
{hook h='displayTop'}
{hook h='displayProductPriceBlock' product=$product type='unit_price'}
```

Remarque : Par convention, et pour bénéficier de la nouvelle interface Widget, le nom d'un hook commence par "display".

La fonction widget

La fonction *widget* permet d'insérer le contenu d'un module développé avec la nouvelle interface widget de PrestaShop 1.7.

Exemple pour ajouter les boutons de partage sur les réseaux sociaux :

```
<div id="product-social-sharing">
    {widget name="socialsharing"}
</div>
```

Le module peut ne pas être installé ni activé. Il est possible de demander au module de renvoyer le contenu d'un hook spécifique avec le paramètre `hook=`

La fonction widget_block

La fonction *widget_block* permet d'insérer le template utilisé par module qui utilise la nouvelle interface widget de PrestaShop 1.7.

Exemple avec le module *ps_linklist* :

```
{widget_block name="ps_linklist"}
    {foreach $linkBlocks as $linkBlock}
        <ul>
            {foreach $linkBlock.links as $link}
                <li><p>
                    <h4>
                        <a href="{ $link.url }">{ $link.title }</a>
                    </h4>
                    <p>{ $link.description }</p>
                </p></li>
            {/foreach}
        </ul>
    {/foreach}
{/widget_block}
```

Le module peut ne pas être installé ni activé.

La fonction l

La fonction *l* est utilisé pour gérer la localisation (traduction) des chaînes de caractères.

Exemple d'utilisation :

```
{l s="Hello" mod="helloworld"}
{l s="Add" d='Admin.Actions'}
{l s='%product_count% item in your cart' sprintf=['%product_count%' => $products_count] d='Shop.Theme.Checkout'}
```

Le paramètre `mod` permet de préciser le module qui contient la traduction (Ancien système de traduction).

Le paramètre `d` permet de préciser le domaine qui contient la traduction (Nouveau système de traduction depuis PrestaShop 1.7.0.0).

La fonction dateformat

La fonction *dateformat* permet de formater une date en fonction de la localisation.

Exemple d'utilisation :

```
{dateFormat date=$order.date_add}
{dateFormat date=$mydate full=1}
```

La fonction debug

La fonction *debug* affiche les variables disponibles dans une popup.

Exemple d'utilisation :

```
{debug}
```

Nouveautés de la 1.7.

Filtres et fonctions supprimés

Filtres et fonctions supprimés sur le front-office dans Smarty :

- secureReferrer
- p (ppp), d (ddd)
- toolsConvertPrice, convertPrice, convertPriceWithCurrency
- displayWtPrice, displayWtPriceWithCurrency, displayPrice, convertAndFormatPrice
- getAdminToken, displayAddressDetail
- getWidthSize, getHeightSize
- addJsDef, addJsDefL

Echappement des variables

Sur PrestaShop 1.6 et sur PrestaShop 1.7 back-office legacy

Exemple de code PHP :

```
$var = '<b>Hello</b>';
$smarty->assign('var', $var);
```

Exemple d'instruction Smarty :

```
{ $var }                { * Hello in Bold (XSS) * }
{ $var|escape:"html":"utf-8" } { * <b>Hello</b> * }
{ $var|cleanHtml }      { * Hello in Bold (Or nothing if js) * }
```

Sur PrestaShop 1.7 front-office

Exemple de code PHP :

```
$var = '<b>Hello</b>';
$smarty->assign('var', $var);
```

Exemple d'instruction Smarty :

```
{ $var }                { * <b>Hello</b> * }
{ $var nofilter }        { * Hello in Bold (XSS) * }
{ $var|cleanHtml nofilter } { * Hello in Bold * }
```

Les objets sont remplacés par des tableaux

Sur PrestaShop 1.6 et sur PrestaShop 1.7 back-office legacy

Exemple de code PHP :

```
$cmsObject = new CMS(1);  
$smarty->assign('cms', $cmsObject);
```

Exemple d'instruction Smarty :

```
{ $cms->content }      { * XSS * }  
{ $cms->delete() }    { * Security * }
```

Sur PrestaShop 1.7 front-office

Exemple de code PHP :

```
$cmsObject = new CMS(1);  
$smarty->assign('cms', new ObjectPresenter($cmsObject));
```

Exemple d'instruction Smarty :

```
{ $cms.content|cleanHtml nofilter }
```

L'héritage et les blocks

La version 1.7 utilise des templates par défaut qui sont réutilisés dans différentes pages.

Pour modifier la présentation d'une information il suffit de redéfinir le block.

Exemple de modification de l'affichage en fonction de la page :

Avant PrestaShop 1.7 (Utilisation de if/else) :

```
{if $page=="index"}  
  { * Show data in index Page * }  
{elseif $page=="category"}  
  { * Show data in Category Page * }  
{else}  
  { * Show data in other Page * }  
{/if}
```

A partir de PrestaShop 1.7 (Utilisation d'héritage et de block):

```
{ * base.tpl *}  
  
{block name="data"}  
  { * Show data in other Page * }  
{/block}  
  
{ * index.tpl *}  
{extends file="base.tpl"}  
  
{block name="data"}  
  { * Show data in index Page * }  
{/block}
```

```
{* category.tpl *}
{extends file="base.tpl"}

}
{block name="data"}
    {* Show data in Category Page *}
{/block}
```

Mise en forme des informations par le code PHP

Sur PrestaShop 1.6 et sur PrestaShop 1.7 back-office legacy, la mise en forme des informations est réalisée dans Smarty.

Exemple de code PHP :

```
$product_price = 10.5;
$smarty->assign(array(
    'price' => $product_price
    'currency' => $this->context->currency
));
```

Exemple d'instruction Smarty :

```
{displayWtPriceWithCurrency price=$price currency=$currency}
```

Sur PrestaShop 1.7 front-office, la mise en forme est réalisée dans le code PHP.

Exemple de code PHP :

```
$product_price = (PriceFormatter)$formatter->format(10.5);
$smarty->assign('price', $product_price);
```

Exemple d'instruction Smarty :

```
{$price}
```

Presenter

De nouvelles classes *Presenter* sont utilisées pour uniformiser la présentation des informations.

Avantages :

- Prise en charge du contexte (Shop / Lang / Currency / Taxes / ...) pour formater les informations.
- Mise à disposition des mêmes informations sur toutes les pages.
- A partir de la Version 1.7.5, lecture des informations "A la demande" (LazyArray).

Les classes 'Presenter'

Les classes *Presenter* dans PrestaShop 1.7.5 :

- ObjectPresenter
- CartPresenter
- ProductListingPresenter
- ProductPresenter
- OrderPresenter

- OrderReturnPresenter
- ModulePresenter

Exercices



Héritage de template

Ne plus afficher les prix des produits mais juste "Promotion" sur la page Promotions.

- Rechercher le nom du block qui contient l'affichage des prix (dans **/catalog/_partials/miniatures/product.tpl**)
- Modifier le fichier **/catalog/listing/prices_drop.tpl**

Solution héritage de template

Comment ne plus afficher les prix des produits mais juste "Promotion" sur la page Promotions.

Solution proposée :

```
{extends file='catalog/listing/product-list.tpl'}

{block name='product_price_and_shipping'}
  <div class="product-price-and-shipping">Promotion</div>
{/block}
```

Cette solution, modifie le block *product_price_and_shipping* dans le template **prices_drop.tpl**

Modification d'une liste

Ajouter le nom de la catégorie par défaut d'un produit, devant son nom, sur la liste des nouveaux produits.

- Rechercher le nom du block qui contient l'affichage du nom du produits (dans **/catalog/_partials/miniatures/product.tpl**)
- Modifier le fichier **/catalog/listing/new-products.tpl**

Solution modification d'une liste

Comment ajouter le nom de la catégorie par défaut d'un produit, devant son nom, sur la liste des nouveaux produits.

Solution proposée :

```
{extends file='catalog/listing/product-list.tpl'}

{block name='product_name'}
    <h2 class="h3 product-title" itemprop="name" style="margin-top:0">
        <a href="{ $product.url }">{ $product.category_name}<br>{ $product.name|t
runcate:30:'...'}</a>
    </h2>
{/block}
```

Cette solution modifie le block *productproduct_name_price_and_shipping* dans le template **new-products.tpl**

Twig

Twig



Twig est un moteur de templates pour le langage de programmation PHP, utilisé par défaut par le framework Symfony.

Moteur de template

Symfony et le composant form utilisent Twig.

Les nouveaux formulaires du back-office de PrestaShop utilisent donc Twig.

Principe de la syntaxe :

```
{{ ... }} affiche
{% ... %} execute
{# ... #} commentaires
```

Syntaxe de base

```
{# this is a comment #}
{{ my_var }} show my_var
{{ my_array['key'] }} ou {{ my_array.key }} show key in my_array .
```

Les variables

Initialisation d'une variable.

```
{% set my_var = "my_value" %}
```

Initialisation de plusieurs variables

```
{% set my_first_var = "my_first_value", my_second_var = "my_second_value" %}
```

Les filtres, syntaxe classique

On peut appliquer des « filtres » sur une variable.

```
{{ my_var|upper }}  
  
{{ my_var|striptags|title }}
```

La fonction filter

Exemple avec l'instruction filter :

```
{% filter upper %}  
uppercase text  
{% endfilter %}  
  
{% filter upper|escape %}  
test in uppercase <em>html visible</em>  
{% endfilter %}
```

Sécurité - XSS

Par défaut, le contenu d'une variable est protégé.

Pour autoriser l'affichage du html vous devez utiliser le modificateur/filtre *raw* :

```
{{ ma_variable_html|raw }}
```

Les tests

Exemple :

```
{% if online > 1 %}  
    There are {{ online }} members online  
{% elseif online == 1 %}  
    There is only one user online (You!)  
{% else %}  
    There are no users online.  
{% endif %}
```

Condition defined

Exemple :

```
{% if session.pseudo is defined %}  
    Your username is {{ session.pseudo }} .  
{% endif %}
```

Condition divisibleby

Exemple :

```
{% if groupe is divisibleby(6) %}  
    {# Create two groups of 6 #}  
{% endif %}
```

Condition empty

Exemple :

```
{% if groupe is empty %}
    {% The group is empty %}
{% endif %}
```

Les boucles

```
{% for i in 0..50 %}
    This is the line {{ i }}
{% endfor %}

{% for lettre in 'a'..'z' %}
    The {{ letter }} is the {{ loop.index }} letter of the alphabet.
{% endfor %}
```

Les boucles avec un tableau

```
{% for product in products %}
    {{ product.name }} : {{ product.price }}
{% endfor %}

{% # only product with description %}
{% for product in products if description is not empty %}
    {{ product.name }} - {{ product.description }}
{% endfor %}
```

La variables loop

A l'intérieur d'une boucle, vous pouvez utiliser *loop* pour avoir des informations.

Exemples :

```
{{ loop.index }} The number of the current iteration (starting with 1).
{{ loop.index0 }} The number of the current iteration (starting with 0).
{{ loop.revindex }} The number of remaining iterations before the end of the
loop (ending with 1).
{{ loop.revindex0 }} The number of remaining iterations before the end of the
loop (ending with 0).
{{ loop.first }} true if this is the first iteration, false otherwise.
{{ loop.last }} true if this is the last iteration, false otherwise.
{{ loop.length }} The total number of iterations in the loop.
```

Les includes

Exemples :

```
{% include 'header.twig' %}

{% include connected ? 'welcome.twig' : 'login.twig' %}
```

Les macros

Exemple de déclaration dans le fichier **forms.html.twig**

```
{% macro input(name, value, type, size) %}
    <input type="{{ type|default('text') }}" name="{{ name }}" value="{{ value|e }}" size="{{ size|default(20) }}" />
{% endmacro %}

{% macro textarea(name, value, rows) %}
    <textarea name="{{ name }}" rows="{{ rows|default(10) }}" cols="{{ cols|default(40) }}">{{ value|e }}</textarea>
{% endmacro %}
```

Exemple d'utilisation :

```
{% import 'forms.html.twig' as forms %}

<dl>
    <dt>Username</dt>
    <dd>{{ forms.input('username') }}</dd>
    <dt>Password</dt>
    <dd>{{ forms.input('password', none, 'password') }}</dd>
</dl>
<p>{{ forms.textarea('comment') }}</p>
```

Les blocs

Exemple de template "Parent" : le fichier **base.html.twig**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    {% block head %}
        <link rel="stylesheet" href="style.css" />
        <title>{{ block title }}{% endblock %} - My Webpage</title>
    {% endblock %}
</head>
<body>
    <div id="content">{{ block content }}{% endblock %}</div>
    <div id="footer">
        {% block footer %}
            &copy; Copyright 2009 by <a href="http://domain.invalid/">you</a>.
        {% endblock %}
    </div>
</body>
</html>
```

L'héritage

Exemple d'héritage.

```
{% extends "base.html.twig" %}

{% block title %}Index{% endblock %}

{% block head %}
    {{ parent() }}
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}

{% block content %}
    <h1>Index</h1>
    <p class="important">
        Welcome on my awesome homepage.
    </p>
{% endblock %}
```

Les extensions ajoutées pour PrestaShop

- Des variables globales
- Des filtres (*Twig_SimpleFilter*)
- Des fonctions (*Twig_SimpleFunction*)

Variables globales

- default_currency
- root_url
- js_translatable

Filtres (*Twig_SimpleFilter*)

- arrayCast
- intCast
- unsetElement
- renderhook
- renderhooksarray
- configuration

Fonctions (*Twig_SimpleFunction*)

- arrayCast
- intCast
- renderhook
- renderhooksarray
- hookcount
- getLegacyLayout
- getAdminLink
- trans
- transchoice

Mécanisme de surcharge

- Pour surcharger les templates du Bundle PrestaShop.
- Créer un répertoire **PrestaShopBundle/views** dans le dossier **/app/Resources** de votre installation.
- Copier les templates du répertoire **src/PrestaShopBundle/Resources/views** dans le nouveau dossier **/app/Resources/PrestaShopBundle/views**

Module

Organisation des modules

Un module est un ensemble de fichiers contenus dans un répertoire **{module_name}** lui-même contenu dans le répertoire **/modules** à la racine du dossier principal de PrestaShop.

Depuis PrestaShop 1.7.0, il existe deux organisations pour les modules :

- L'organisation *legacy* qui permet à un module de fonctionner sur PrestaShop 1.6 et PrestaShop 1.7.
- La nouvelle organisation *moderne* pour bénéficier des derniers ajouts de PrestaShop 1.7.

Module legacy

Les répertoires dans un module legacy :

- **/modules/{module_name}**
 - **/classes** les classes du module
 - **/controllers** les contrôleurs du module
 - **/override** les surcharges (avec installation et désinstallation automatique)
 - **/translations** les fichiers de traduction
 - **/upgrade** les fichiers pour les mises à jour
 - **/views** les divers fichiers de vue (JavaScript, Templates, CSS, etc.)
 - **{module_name}.php** le fichier « d'accroche »
 - **index.php**
 - **logo.png** le logo 128x128 du module
 - **Readme.md** le fichier readme du module

Module moderne

Les répertoires dans un module moderne :

- **/modules/{module_name}**
 - **/config** le répertoire configuration
 - **/controllers** les contrôleurs front-office du module
 - **/src** les classes du module
 - **/translations** les fichiers de traduction
 - **/upgrade** les fichiers pour les mises à jour
 - **/vendor** les librairies utilisées par le module
 - **/views** les divers fichiers de vue (JavaScript, Templates, CSS, etc.)
 - **{module_name}.php** le fichier principal du module
 - **composer.json**
 - **index.php**
 - **logo.png** le logo 128x128 du module
 - **Readme.md** le fichier readme du module
 - **self_config.yml** pour configurer le module par ligne de commande

Remarque : Les répertoires **/classes** et **/override** ne doivent plus être utilisées.

Fichiers de cache

Des répertoires et des fichiers de cache peuvent être créés par PrestaShop, Composer ou npm dans un module.

Remarque : Un module ne doit pas stocker de données en cache dans le répertoire `/modules/{module_name}`. Pour stocker des informations en cache, un module doit utiliser le mécanisme de cache de *Smarty ObjectModel* ou le répertoire cache de PrestaShop `_PS_CACHE_DIR_`

- `/modules/{module_name}`
 - **composer.lock** fichier lock de composer
 - **config.xml** le fichier cache de configuration
 - **config_fr.xml** le fichier cache de configuration en français
 - **/views**
 - **/_dev**
 - **package-lock.json** fichier lock de npm
 - **/node_modules** répertoire créer par npm

Avertissement : Le répertoire `/modules/{module_name}/views/_dev/node_modules` NE DOIT JAMAIS ETRE COPIE sur un serveur en production.

Le fichier principal du module

Le fichier `/modules/{module_name}/{module_name}.php` est obligatoire. Il contient la classe qui permet d'accrocher le code du module au coeur de PrestaShop.

En fonction du type de module, la classe héritera de :

- *CarrierModule* pour les modules de transports,
- *PaymentModule* pour les modules de paiements,
- *TaxManagerModule* pour les modules qui proposent de nouvelles règles de taxes,
- *ModuleGrid* pour les modules qui ajoutent des tableaux statistiques,
- *ModuleGraph* pour les modules qui ajoutent des graphiques statistiques,
- *Module* pour tous les autres modules.

L'interface WidgetInterface

Depuis PrestaShop 1.7, il est possible d'implémenter l'interface *WidgetInterface*

Cette interface, permet d'utiliser les instructions `{widget name="..."}` et `{widget_block name="..."}`...`{/widget_block}` pour appeler le module directement depuis un template *Smarty*. L'utilisation de la *WidgetInterface* ne vérifie pas si le module est actif ou même installé.

Exemple d'utilisation :


```

<?php

use PrestaShop\PrestaShop\Core\Module\WidgetInterface;

class MySample extends Module implements WidgetInterface
{
    // ....
    public function renderWidget($hookName, array $params)
    {
        $this->smarty->assign(
            $this->getWidgetVariables($hookName, $params)
        );
        return $this->fetch(
            'module:mysample/views/templates/widget/mysample.tpl'
        );
    }

    public function getWidgetVariables($hookName, array $params)
    {
        return array(
            'var_name' => 'the value'
        );
    }
}

```

Module legacy

Le répertoire classes

Le répertoire **/modules/{module_name}/classes** contient les classes utilisées par la module.

Avertissement : Les fichiers dans ce répertoire n'utilise pas de namespace. Le chargement du code dans le module est réalisé via des instructions **include_once** .

- **/modules/{module_name}/classes**
 - **index.php**
 - **mymodel.php**
 - **mytools.php**
 - ...

Conseil : Avec PrestaShop 1.7 et l'utilisation de composer (PHP 5.4+), il est possible d'automatiser le chargement des classes en ajoutant le répertoire **classes** dans le tableau **autoload.classmap** du fichier **composer.json** du module.

Le répertoire controllers

Le répertoire **/modules/{module_name}/controllers** contient les contrôleurs front-office et back-office legacy du module.

Avertissement : Les contrôleurs dans ce répertoire n'utilise pas de namespace. Le chargement des contrôleurs dans le module est réalisé via des instructions **include_once** .

- **/modules/{module_name}/controllers**
 - **/admin** les contrôleurs back-office
 - **admin{module_name}Controller.php**
 - **index.php**
 - **/front** les contrôleurs front-office

- **default.php**
 - **index.php**
- **index.php**

Conseil : Avec PrestaShop 1.7 et l'utilisation de composer (PHP 5.4+), il est possible d'automatiser le chargement des contrôleurs en ajoutant le répertoire **classes** dans le tableau **autoload.classmap** du fichier **composer.json** du module.

Le répertoire translations

Le répertoire **/modules/{module_name}/translations** contient les traductions legacy du module.

- **/modules/{module_name}/translations**
 - **en.php**
 - **es.php**
 - **fr.php**
 - **index.php**

Conseil : Le répertoire **translations** d'un module peut être copié dans le répertoire **/themes/{active_theme}/modules/{module_name}/translations** pour adapter les traductions du module au thème actif.

Le répertoire upgrade

Le répertoire **/modules/{module_name}/upgrade** contient les instructions exécutés lors de la mise à jour du module.

- **/modules/{module_name}/upgrade**
 - **index.php**
 - **install-1.1.php**
 - **install-2.1.php**
 - **install-2.2.php**

Remarque : Pour mettre à jour ce module de la version 1.1.1 à la version 2.2.0, PrestaShop va exécuter les fonctions **upgrade_module_2_1(\$module)** et **upgrade_module_2_2(\$module)** des fichiers **install-2.1.php** et **install-2.2.php**

Le répertoire views

Le répertoire **/modules/{module_name}/views** contient les fichiers de vue (JavaScript, Templates, CSS, etc.) du module.

- **/modules/{module_name}/views**
 - **/css**
 - **index.php**
 - **/js**
 - **/sass**
 - **/templates**
 - **/admin**
 - **/front**
 - **/hooks**
 - **index.php**

Conseil : Le répertoire **views** d'un module peut être copié dans le répertoire

`/themes/{active_theme}/modules/{module_name}/views` pour adapter la partie visuel du module au thème actif.

Création d'un module legacy

Création d'un module

Nous allons créer un module hello world.

Pour commencer, vous devez créer le répertoire **helloworld** et le fichier **helloworld/helloworld.php** dans le répertoire **modules** de PrestaShop.

- **modules/helloworld**
 - **helloworld.php**

Création du fichier d'accroche.

Le fichier **helloworld.php** permet d'accrocher notre module à PrestaShop.

```
<?php

if (!defined('_PS_VERSION_')) {
    exit;
}

use PrestaShop\PrestaShop\Core\Module\WidgetInterface;

class HelloWorld extends Module implements WidgetInterface
{
    public function __construct()
    {
        $this->name = 'helloworld';
        $this->version = '1.0.0';
        $this->author = 'Me';
        $this->bootstrap = true;
        parent::__construct();

        $this->displayName = $this->l('Hello World');
        $this->description = $this->l('This is an example module');
    }
}
```

Options

```
$this->tab = 'front_office_features';
$this->ps_versions_compliancy = array('min' => '1.5', 'max' => '1.6');
$this->dependencies = array('blockcart'); // need blockcart module.
$this->need_instance = false;
$this->limited_countries = array('fr');
$this->is_configurable = false;
$this->controllers = array('default');
```

Les différentes valeurs possibles pour tab

- **administration** : Administration
- **advertising_marketing** : Advertising & Marketing

- analytics_stats : Analytics & Stats
- billing_invoicing : Billing & Invoices
- checkout : Checkout
- content_management : Content Management
- export : Export
- front_office_features : front-office Features
- i18n_localization : I18n & Localization
- market_place : Market Place
- merchandizing : Merchandizing
- migration_tools : Migration Tools
- others : Other Modules
- payments_gateways : Payments & Gateways
- payment_security : Payment Security
- pricing_promotion : Pricing & Promotion
- quick_bulk_update : Quick / Bulk update
- search_filter : Search & Filter
- seo : SEO
- shipping_logistics : Shipping & Logistics
- slideshows : Slideshows
- smart_shopping : Smart Shopping
- social_networks : Social Networks

La fonction install()

Cette fonction permet de contrôler l'installation du module depuis l'administration des modules.

```
<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function install()
    {
        if (Shop::isFeatureActive()) {
            Shop::setContext(Shop::CONTEXT_ALL);
        }

        if (!parent::install() ||
            !$this->registerHook('displayLeftColumn') ||
            !$this->registerHook('displayHeader') ||
            !Configuration::updateValue('HELLO_WORLD_NAME', 'World')) {
            return false;
        }
        return true;
    }
}
```

La fonction uninstall()

Cette fonction permet de contrôler la désinstallation du module depuis l'administration.

```
<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function uninstall()
    {
        if (!parent::uninstall() ||
            !Configuration::deleteByName('HELLO_WORLD_NAME')
        ) {
            return false;
        }
        return true;
    }
}
```

Vous devez :

- Supprimer les valeurs ajoutées dans la table configuration (obligatoire).
- Supprimer les tables ajoutées dans la base de données (non obligatoire).

WidgetInterface

Les fonctions `renderWidget` et `getWidgetVariables`.

```
<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function renderWidget($hookName, array $params)
    {
        $this->smarty->assign(
            $this->getWidgetVariables($hookName, $params)
        );
        return $this->fetch(
            'module:helloworld/views/templates/hook/helloworld.tpl'
        );
    }

    public function getWidgetVariables($hookName, array $params)
    {
        return array(
            'name' => Configuration::get('HELLO_WORLD_NAME')
        );
    }
}
```

Prendre en charge les Hooks

Pour que notre module puisse afficher quelque chose sur le site, nous avons besoin d'ajouter les fonctions qui vont prendre en charge l'affichage.

La fonction `hookDisplayLeftColumn()` permet de traiter le hook `displayLeftColumn` :

```
<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....
    public function hookDisplayLeftColumn($params)
    {
        $this->context->smarty->assign(array(
            'name' => Configuration::get('HELLO_WORLD_NAME')
        ));
        return $this->fetch(
            'module:helloworld/views/templates/hook/helloworld.tpl'
        );
    }
}
```

La fonction *hookDisplayHeader()* permet de traiter le hook *displayHeader* :

```
<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function hookDisplayHeader($params)
    {
        $this->context->controller->registerStylesheet(
            'modules-helloworld',
            'modules/'.$this->name.'/views/css/helloworld.css',
            ['media' => 'all', 'priority' => 150]
        );
        // $this->context->controller->registerJavascript(
        //     'modules-helloworld',
        //     'modules/'.$this->name.'/views/js/helloworld.js',
        //     ['position' => 'bottom', 'priority' => 150]
        // );
    }
}
```

Remarque : Il est possible d'utiliser l'interface *WidgetInterface* pour traiter tous les hooks avec la même fonction.

Le fichier template du module

Le fichier */views/templates/hook/helloworld.tpl*

```
{block name='hello world'}
<div class="hello-world">
  <h4>{l s='Message' mod='helloworld'}</h4>
  <p>
    {l s='Hello %s.' sprintf=[$name] mod='helloworld'}
  </p>
</div>
{/block}
```

Le fichier CSS de notre module

Le fichier */views/css/helloworld.css*

```
.hello-world p {  
    font-weight: bold;  
}
```

Ajouter une page de configuration

Pour rendre le module configurable (Ajout automatique du lien 'Configurer' dans l'administration), il faut ajouter la fonction *getContent()* à notre module.

```
<?php  
  
class HelloWorld extends Module implements WidgetInterface  
{  
    // ....  
  
    public function getContent()  
    {  
        $output = '';  
        if (Tools::isSubmit('submit'.$this->name)) {  
            $helloWorldName = Tools::getValue('HELLO_WORLD_NAME');  
            if (!$helloWorldName  
                || empty($helloWorldName)  
                || !Validate::isGenericName($helloWorldName)) {  
                $output .= $this->displayError(  
                    $this->l('Invalid Configuration value')  
                );  
            } else {  
                Configuration::updateValue(  
                    'HELLO_WORLD_NAME',  
                    $helloWorldName  
                );  
                $output .= $this->displayConfirmation(  
                    $this->l('Settings updated')  
                );  
            }  
        }  
        return $output.$this->displayForm();  
    }  
}
```

Afficher la page de configuration

Par convention, la fonction *displayForm()* est utilisée pour afficher la page de configuration d'un module.

```

<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function displayForm()
    {
        // Get default Language
        $default_lang = (int)Configuration::get('PS_LANG_DEFAULT');

        // Init Fields form array
        $fields_form = array();
        $fields_form[0]['form'] = array(
            'legend' => array(
                'title' => $this->l('Hello settings'),
            ),
            'input' => array(
                array(
                    'type' => 'text',
                    'label' => $this->l('Configuration value'),
                    'name' => 'HELLO_WORLD_NAME',
                    'size' => 20,
                    'required' => true
                ),
            ),
            'submit' => array(
                'title' => $this->l('Save')
            )
        );

        $helper = new HelperForm();
        // Module, Token and currentIndex
        $helper->module = $this;
        $helper->name_controller = $this->name;
        $helper->token = Tools::getAdminTokenLite('AdminModules');
        $helper->currentIndex = AdminController::$currentIndex
            . '&configure='
            . $this->name;
        // Language
        $helper->default_form_language = $default_lang;
        $helper->allow_employee_form_lang = $default_lang;
        // title and Submit
        $helper->title = $this->displayName;
        $helper->submit_action = 'submit'.$this->name;
        // Load current value
        $helper->fields_value['HELLO_WORLD_NAME'] =
            Configuration::get('HELLO_WORLD_NAME');

        return $helper->generateForm($fields_form);
    }
}

```

Création d'un contrôleur back-office

Création d'un contrôleur back-office

Un contrôleur back-office legacy hérite de la classe *ModuleAdminController* :

- Le fichier source est dans le répertoire **/controllers/admin** du module.
- Le template est dans le répertoire **/views/templates/admin** du module.

Exemple de contrôleur back-office

Exemple de contrôleur :

- Le fichier source d'un contrôleur back-office est dans le répertoire **/controllers/admin** d'un module.
- Par convention, le nom du fichier commence par Admin.
- Le nom de la classe doit finir par Controller.

Fichier source du contrôleur "AdminHelloworld"

/modules/helloworld/controllers/admin/adminhelloworldController.php

```
<?php

class AdminHelloworldController extends ModuleAdminController
{
    public function __construct()
    {
        $this->bootstrap = true;
        $this->display = 'view';
        parent::__construct();

        $this->context->smarty->assign(array(
            'name' => Configuration::get('HELLO_WORLD_NAME')
        ));
    }
}
```

La fonction d'installation

L'installation du contrôleur dans le menu du back-office nécessite la création d'une fonction spécifique.

```
<?php

class AdminHelloworldController extends ModuleAdminController
{
    // ...
    public static function installInBO($module, $name)
    {
        $new_menu = new Tab();
        $new_menu->id_parent = Tab::getIdFromClassName('AdminCatalog');
        $new_menu->class_name = 'AdminHelloworld';
        $new_menu->module = $module->name;
        $new_menu->active = 1;

        // Set menu name in all active Language.
        $languages = Language::getLanguages(true);
        foreach ($languages as $language) {
            $new_menu->name[(int)$language['id_lang']] = $name;
        }

        return $new_menu->save();
    }
}
```

La fonction de désinstallation

La suppression du contrôleur du menu du back-office nécessite la création d'une fonction de désinstallation.

```
<?php

class AdminHelloworldController extends ModuleAdminController
{
    // ...
    public static function removeFromBO()
    {
        $remove_id = Tab::getIdFromClassName('AdminHelloworld');
        if ($remove_id) {
            $to_remove = new Tab((int)$remove_id);
            if (Validate::isLoadedObject($to_remove)) {
                return $to_remove->delete();
            }
        }
        return false;
    }
}
```

Le fichier template du contrôleur back-office

Fichier template du contrôleur "AdminHelloworld"

/modules/helloworld/views/templates/admin/helloworld/helpers/view/view.tpl :

```
<h1>{l s='Welcome in Admin Hello World' mod='helloworld'}</h1>
<p>{l s='Hello "%s".' sprintf=[$name] mod='helloworld'}</p>
```

Modification du module

Le module doit installer et désinstaller automatiquement le contrôleur dans le menu du back-office de PrestaShop.

La fonction d'installation du module doit appeler la fonction d'installation du contrôleur :

```

<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function install()
    {
        include_once dirname(__FILE__)
            .'/controllers/admin/adminhelloworldController.php';

        if (Shop::isFeatureActive()) {
            Shop::setContext(Shop::CONTEXT_ALL);
        }

        if (!parent::install() ||
            !AdminHelloworldController::installInBO(
                $this,
                $this->l('Admin Hello World')
            )
            || !$this->registerHook('displayLeftColumn') ||
            !$this->registerHook('displayHeader') ||
            !Configuration::updateValue('HELLO_WORLD_NAME', 'World')) {
            return false;
        }
        return true;
    }
}

```

La fonction de désinstallation du module doit appeler la fonction de désinstallation du contrôleur :

```

<?php

class HelloWorld extends Module implements WidgetInterface
{
    // ....

    public function uninstall()
    {
        include_once dirname(__FILE__)
            .'/controllers/admin/adminhelloworldController.php';

        if (!parent::uninstall() ||
            !AdminHelloworldController::removeFromBO() ||
            !Configuration::deleteByName('HELLO_WORLD_NAME')
        ) {
            return false;
        }
        return true;
    }
}

```

Ajax

Principe d'Ajax

Ajax (acronyme d'Asynchronous JavaScript and XML) permet de construire des sites web dynamiques.

L'idée même d'AJAX est de faire communiquer une page Web avec un serveur Web sans occasionner le rechargement de la page.

Remarque : Les demandes en Ajax sont effectuées de manière asynchrone : le navigateur Web continue d'exécuter le JavaScript alors que la demande est partie, il n'attend pas la réponse envoyée par le serveur Web.

La fonction ajax de jQuery

`jQuery.ajax(url [, settings])` permet d'appeler un script ajax.

- url : URL à laquelle envoyer la requête.
- settings : paramètres et fonctions

Paramètres et fonctions

Quelques paramètres :

- async (boolean) : par défaut true
- type : par défaut 'GET'
- cache (boolean) : par défaut true (false pour dataType 'script' and 'jsonp') - utilisation du cache du navigateur ?
- data : tableau envoyé au serveur
- dataType : (xml, json, script, or html)

Quelques fonctions :

- beforeSend : exécuté avant le l'appel ajax
- error : en cas d'erreur
- success : en cas de succès
- timeout : en cas de timeout
- complete : exécution après l'appel a success ou error, quelque soit le résultat.

La fonction jQuery.load

`jQuery.load(url [, data] [, complete(responseText, textStatus, XMLHttpRequest)])` permet de charger directement le résultat HTML d'un appel ajax dans un élément de la page.

Exemple :

```
$('#result').load('ajax/test.php', function() {  
    alert('Load was performed.');
```

La fonction jQuery.get

`jQuery.get(url [, data] [, success(data, textStatus, jqXHR)] [, dataType])` permet d'appeler un script ajax avec une requête HTTP GET.

Elle est équivalente à l'appel ajax suivant :

```
$.ajax({  
    url: url,  
    data: data,  
    success: success,  
    dataType: dataType  
});
```

Valeur de retour sur un appel ajax avec jQuery

Toutes ces fonctions jQuery retournent un objet **jqXHR** , une extension de **XMLHttpRequest** .

Cet objet permet de gérer des fonctions suivant le résultat de l'appel.

- Success : **jqXHR.done(function(data, textStatus, jqXHR) {});**
- Error : **jqXHR.fail(function(jqXHR, textStatus, errorThrown) {});**
- Always : **jqXHR.always(function(data|jqXHR, textStatus, jqXHR|errorThrown) { });**

Exemple avec appel de fonction suivant le résultat de la requête.

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.get("example.php", function() {
    alert("success");
})
.done(function() { alert("second success"); })
.fail(function() { alert("error"); })
.always(function() { alert("finished"); });
// perform other work here ...
// Set another completion function for the request above
jqxhr.always(function(){ alert("second finished"); });
```

Utilisation d'Ajax dans un Contrôleur back-office

Lorsque PrestaShop détecte la présence du paramètre "ajax" dans la requête (GET ou POST), la propriété ajax du contrôleur devient vrai (\$this->ajax = true).

Le fonctionnement du contrôleur est alors modifié pour permettre l'appel automatique d'une méthode en fonction du paramètre "action" de la requête.

Exemple, pour l'url `http://.....&ajax&action=reset`

Le contrôleur va exécuter, si elle existe la méthode *displayAjaxReset()* , sinon la fonction *displayAjax()* (Par défaut).

```
<?php

class AdminHelloworldController extends ModuleAdminController
{
    // ....
    public function displayAjaxReset()
    {
        // ?ajax=1&action=reset
        die('Ajax:Reset');
    }

    public function displayAjax()
    {
        // ?ajax=1&action!=reset
        die('Ajax:OK');
    }
}
```

Override de module

Override de module

PrestaShop propose un mécanisme d'override pour les modules.

Exemple pour la classe *MyModule* créer dans le fichier **/modules/mymodule/mymodule.php** . Vous pouvez surcharger la classe d'un module en créant une classe *MyModuleOverride* dans le fichier **/override/modules/mymodule/mymodule.php**

Remarque : Le mécanisme d'override n'est possible que pour la classe principale du Module.

Exemple d'override

Exemple pour le module HelloWorld :

```
<?php
use PrestaShop\PrestaShop\Core\Module\WidgetInterface;

class HelloWorldOverride extends HelloWorld implements WidgetInterface
{
    public function __construct()
    {
        parent::__construct();
        $this->displayName = $this->l('Hello World (Override)');
        $this->description = $this->l('This is an example module (Override)');
    }
}
```

Remarque : Pour activer les nouveaux overrides, vous devez vider le cache. Rendez-vous dans le back-office, **Paramètres avancés > Performances** , Vider le cache

Quand faire un override de module ?

L'override de module n'est pas conseillé. Cette technique doit être réservée pour des modifications très légères sur la classe du module.

Remarque : Pour adapter le fonctionnement d'un module à une boutique, il est préférable de copier, renommer le module (Attention à bien changer toutes les occurrences du nom.) et reprendre complètement la maintenance et l'évolution du module.

Mise à jour de module

Principes

Il existe un mécanisme permettant de gérer la mise à jour d'un module.

Pour mettre à jour un module, il faut simplement remplacer les fichiers du module par ceux de la nouvelle version.

PrestaShop détecte la présence de la mise à jour en comparant la version du module **\$this->version** et la dernière version du module exécutée (L'information est dans la table module.).

Le module apparaît dans la liste des modules à mettre à jour **Modules > Module manager > Updates**

Mise à jour d'un module

Lors de la mise à jour d'un module :

- PrestaShop charge tous les fichiers **install-{major}-{minor}.php** du répertoire

/modules/{module_name}/upgrade pour la nouvelle version.

- PrestaShop exécute les fonctions **upgrade_module_{major}_{minor}(\$module)** entre la dernière version exécutée et la nouvelle version, si les fonctions existent.

Remarque : Une fonction *upgrade_module_{major}_{minor}(\$module)* ne peut retourner que **true** ou **false**.

Exemple de mise à jour de module

Pour mettre à jour le module {module_name} de la version 1.1.1 à la version 2.2.0 :

Si nous avons dans le répertoire **/modules/{module_name}/upgrade** les fichiers suivants :

- **/modules/{module_name}/upgrade**
 - **index.php**
 - **install-1.1.php**
 - **install-2.1.php**
 - **install-2.2.php**

Pour mettre à jour le module, PrestaShop va exécuter les fonctions :

- *upgrade_module_2_1(\$module)* du fichier **install-2.1.php**
- *upgrade_module_2_2(\$module)* du fichier **install-2.2.php**

Sécurité

Règles élémentaires

- Mettre à jour PHP et optimiser la configuration du serveur.
- Contrôler les droits d'écritures et d'exécutions sur les répertoires.
- Faire des sauvegardes régulières.
- Mettre à jour la boutique.

Optimiser votre fichier .htaccess

Bloquer tout ce qui n'est pas nécessaire au bon fonctionnement de la boutique.

```
RedirectMatch gone ^/_vti.*
RedirectMatch gone ^/MSOffice.*
RedirectMatch gone ^[-_a-z0-9/\.]*/.*
RedirectMatch gone ^.*etc/passwd.*

# ...

RewriteCond %{REQUEST_METHOD} (GET|POST) [NC]
RewriteCond %{QUERY_STRING} ^(.*) (%3C|<)/?script(.*)$ [NC,OR]
RewriteCond %{QUERY_STRING} ^(.*) (%3D|=)?javascript(%3A|:)(.*)$ [NC,OR]
RewriteCond %{QUERY_STRING} ^(.*)document\.location\.href(.*)$ [OR]
RewriteCond %{QUERY_STRING} ^(.*)base64_encode(.*)$ [OR]
RewriteCond %{QUERY_STRING} ^(.*)GLOBALS(=| [|%[0-9A-Z]{0,2})(.*)$ [OR]
RewriteCond %{QUERY_STRING} ^(.*)_REQUEST(=| [|%[0-9A-Z]{0,2})(.*)$ [OR]
# ...
RewriteRule (.) - [F]
```

Sécuriser votre installation

- Penser à utiliser un login et mot de passe toujours différents.
- N'utilisez pas le préfix 'ps' sur les bases de données en production.
- N'affichez pas sur la boutique, une adresse email utilisée par un employé pour se connecter au back-office.
- Toujours du https pour le back-office.
- Remarque :** N'UTILISEZ JAMAIS LE MÊME MOT DE PASSE pour le FTP, la base de données, la boîte mail, l'interface d'administration du site web, etc.

Manipulation des données

- Ne jamais faire confiance aux données venant de l'extérieur.
- Garder à l'esprit que rien de ce que vous utilisez n'est fiable.
- Dans le domaine de la sécurité il ne faut pas viser plus bas que la perfection.

Never trust foreign data

Toute variable qui n'a pas été assignée au sein de la fonction dans laquelle elle est utilisée doit être considérée comme étrangère.

Une donnée peut être une variable, mais aussi un fichier, un tableau de paramètre (POST, GET, SERVER, etc.), la réponse d'un web service...

Vous devez utiliser les fonctions de la classe Tools :

```
<?php
Tools::getValue($value);
Tools::getIsset($value);
Tools::safeOutput($msg);
//...
```

Contrôler vos données

Si vous attendez un entier, vérifiez que c'est bien un entier.

Utiliser des expressions régulières (Attention à la charge CPU).

Privilégier l'utilisation des fonctions `is_numeric()`, `is_array()`, `is_object()`, etc.

Faites des casts pour être sûr du type de donnée.

Vous devez utiliser les fonctions de la classe Validate :

```
<?php
Validate::isEmail($email);
Validate::isFloat($float);
Validate::isMailName($mail_name);
Validate::isPrice($price);
//...
```

Injectons SQL

En insérant des données dans une requête SQL, il est facile de modifier son comportement.

Un exemple classique :

```
SELECT id FROM users
WHERE email = "email@shop.com"
AND password = "$password"
```

L'injection de `"OR "1" = "1"` dans la variable `$password` donnera la requête suivante :

```
SELECT id FROM users
WHERE email = "email@shop.com"
AND password = "" OR "1" = "1"
```

Il existe deux types d'injections courantes :

- Celles qui exploitent une absence de protection des quotes et doubles quotes
- Celles qui exploitent l'absence de contrôle des types de données

SQL - Se protéger

Pour bénéficier de la meilleure protection, vous devez utiliser les fonctions `pSQL()` et `bqSQL()`.

Pour se protéger des injections qui exploitent l'absence de contrôle des types de données et qui concernent principalement les données de type numérique, vous devez utiliser un cast.

Exemple

```
<?php
$query = 'SELECT id FROM users
WHERE id_user = ' . (int)$id_user . '
AND password = "' . pSQL($password) . '"';
```

XSS

Une faille XSS, consiste à exploiter l'interprétation HTML / JavaScript par le navigateur lors de l'affichage des données.

Exemple d'une page non sécurisée :

```
<html>
<body>
  No results were found for your keyword <?php echo $keyword; ?>.
</body>
</html>
```

Pour exploiter la faille, un pirate peut simplement utiliser le mot clé script

```
<script>alert('Faille XSS');</script>
```

Et un message d'alerte apparaîtra sur la page.

On peut bien évidemment faire bien plus en JavaScript qu'un message d'alerte : redirection vers un autre site, vol de cookies...

XSS - Se protéger

Vous devez utiliser les fonctions de la classe Tools.

```
<?php
$id_product = Tools::getValue('ID');
Tools::safeOutput($string, $html = false);
Tools::htmlentitiesDecodeUTF8($string);
```

Vous devez gérer l'échappement des variables dans vos templates.

```

{*
    ma_var = '<b>Hello</b>';
*}

{* PrestaShop 1.6 *}
{$ma_var}    {* Show Hello in Bold ---> XSS Security risk *}
{$ma_var|escape:"html":"utf-8"}    {* Show <b>Hello</b> *}
{$ma_var|cleanHtml}    {* Show Hello in Bold *}

{* PrestaShop 1.7 *}
{$ma_var}    {* Show <b>Hello</b> *}
{$ma_var|cleanHtml nofilter}    {* Show Hello in bold *}

```

CSRF

Une faille CSRF (Cross-Site Request Forgery) consiste à exploiter l'identité d'un utilisateur autorisé en forçant le navigateur à envoyer une requête à son insu.

Paul est l'administrateur d'un forum sur lequel il est connecté (par sessions ou cookie).

Pierre veut devenir super administrateur du forum :

- Pierre envoie un message à Paul avec une pseudo-image qui est en fait un lien permettant de devenir super administrateur.
- Lorsque Paul ouvre le message et demande à afficher l'image, le script est exécuté à partir de la session de Paul.
- Paul n'a pas connaissance de l'exécution du script.

CSRF - Se protéger

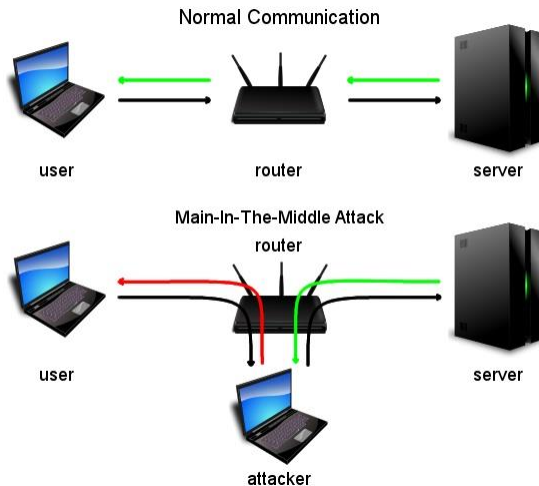
La solution passe par l'utilisation d'un jeton de sécurité, comme vous pouvez le voir dans PrestaShop ou phpMyAdmin.

jeton=: hash avec identifiant d'utilisateur + adresse de la page + un "salt" généré à l'installation.

Pour chaque page ou même chaque action, un jeton unique doit être généré.

Pour toutes actions critiques, demander une confirmation à l'utilisateur.

Man in the middle



L'attaque de l'homme du milieu (HDM) ou man-in-the-middle attack (MITM), parfois appelée attaque de l'intercepteur, est une attaque qui a pour but d'intercepter les communications entre deux parties

Le but de l'attaquant est de se faire passer pour l'un (voire les 2) correspondants, en utilisant, par exemple :

- l'imposture ARP (ARP Spoofing) : c'est probablement le cas le plus fréquent. Si l'un des interlocuteurs et l'attaquant se trouvent sur le même réseau local, il est possible, voire relativement aisé, pour l'attaquant de forcer les communications à transiter par son ordinateur en se faisant passer pour un « relais » (routeur, passerelle) indispensable. Il est alors assez simple de modifier ces communications.
- l'empoisonnement DNS (DNS Poisoning) : L'attaquant altère le ou les serveur(s) DNS des parties de façon à rediriger vers lui leurs communications sans qu'elles s'en aperçoivent.
- l'analyse de trafic afin de visualiser d'éventuelles transmissions non chiffrées.
- le déni de service : l'attaquant peut par exemple bloquer toutes les communications avant d'attaquer un parti. L'ordinateur ne peut donc plus répondre et l'attaquant a la possibilité de prendre sa place.

MITM Se protéger

Il existe différents moyens pour se prémunir contre cette attaque :

- obtenir la clé publique de son interlocuteur par un tiers de confiance. Si les deux interlocuteurs possèdent un contact en commun (le tiers de confiance) alors ce dernier peut servir d'intermédiaire pour transmettre les clés. Les infrastructures à clés publiques sont des systèmes ou des organismes qui permettent de vérifier la validité des clés en se basant principalement sur des certificats.
- échanger les clés par un moyen qui ne permet pas cette attaque : en main propre, par téléphone, etc.
- vérifier le niveau de confiance qui a été accordée à la clé que l'on a en sa possession : certains logiciels comme GnuPG proposent de mettre la clé publique en ligne sur un serveur. Sur ce serveur, d'autres utilisateurs peuvent faire connaître le degré de confiance qu'ils accordent à une clé. On obtient ainsi un graphe qui relie les différents utilisateurs.
- authentification avec un mot de passe ou autre système avancé, comme la reconnaissance vocale ou biologique.

Internationalisation

Principe de base

Contenu en plusieurs langues

PrestaShop permet de saisir des informations en plusieurs langues.

En particulier :

- les fiches produits
- les catégories
- les marques
- les pages cms
- les URL, les balises titres, les meta description...
- le texte de la page d'accueil
- etc.

Pack de localisation

Un pack de localisation permet de configurer PrestaShop pour un pays.

Il contient les informations suivantes :

- États
- Taxes et groupe de Taxes
- Devises
- Langues
- Unités : Poids, dimension, volume, distance

Vous pouvez charger un pack de localisation dans le menu **International > Localisation**

Les packs de localisation sont dans le répertoire **/localization** de PrestaShop.

Exemple de pack

Exemple de pack pour la Tunisie :

```
<?xml version="1.0"?>
<localizationPack name="Tunisia">
  <currencies>
    <currency name="Tunisian Dinar" iso_code="TND" iso_code_num="788" sign="D
T" blank="0" conversion_rate="1.000000" format="2" decimals="1"/>
  </currencies>
  <languages>
    <language iso_code="ar"/>
    <language iso_code="fr"/>
  </languages>
  <taxes>
    <taxRulesGroup name="TVA Tunisie">
      <taxRule iso_code_country="tn" behavior="0" id_tax="55"/>
    </taxRulesGroup>
    <tax id="55" name="TVA" rate="19.000"/>
  </taxes>
  <units>
    <unit type="weight" value="kg"/>
    <unit type="volume" value="L"/>
    <unit type="short_distance" value="cm"/>
    <unit type="base_distance" value="m"/>
    <unit type="long_distance" value="km"/>
  </units>
</localizationPack>
```

Taxes

Pour paramétrer les taxes, PrestaShop utilise deux informations, les taxes et les règles de taxes.

- Une taxe correspond à un nom et à un taux.
- Une règle de taxe permet d'appliquer ou non une taxe à un ensemble de pays, d'état, de région ou de code postaux.

Les taxes sont paramétrées depuis le menu **International > Taxes**

Il est possible de préciser le comportement en cas d'adresse correspondant à plusieurs règles de taxes.

- Cette taxe uniquement : aucune des autres taxes correspondant au client ne sera appliquée.
- Combiner les taxes : exemple, 100€ + (10% + 5% => 15%) => 115€.
- L'une après l'autre : exemple, 100€ + 10% => 110€ + 5% => 115,50€.

Remarque : Les taxes sont appliquées par défaut en fonction de l'adresse de livraison. Vous pouvez choisir l'adresse de facturation en modifiant les options de taxes.

Transporteurs

La configuration des transporteurs utilise la notion de zone de transport.

En fonction d'une zone de transport, il est possible d'activer ou non un transporteur et de paramétrer les frais de livraison.

Une zone peut être un continent, un pays, un groupe de pays, une région, une ville, etc...

Les zones sont paramétrées depuis le menu **International > Zones géographiques**

Moyen de paiement

Il est possible de désactiver un moyen de paiement :

- en fonction du pays

- en fonction de la devise
- en fonction du transporteur
- en fonction du groupe de client

Pour configurer les moyens de paiement, rendez-vous dans le menu **Paiement > Préférences**

Traductions

Toutes les traductions sont modifiables à partir du menu **International > Traductions**

Les traductions sont regroupées par grandes catégories :

- Traductions du back-office pour la partie administration de la boutique.
- Traductions du thème pour la partie front-office (modules, thèmes, notifications, PDF, etc.).
- Traductions des modules installés.
- Traductions des emails pour les sujets et le corps des messages.
- Autres traductions pour les textes provenant de l'ancien thème legacy du back-office.

Themes

Vous pouvez proposer un thème avec des templates différents en fonction de la langue.

Lors de l'utilisation d'un template, PrestaShop vérifiera de nombreux emplacements pour déterminer quel fichier doit être utilisé. Il utilise le code iso de la langue courante dans cette recherche.

Pour afficher un template (Par exemple product.tpl) en français, PrestaShop va rechercher le template à utiliser dans l'ordre suivant :

- **/themes/{mon theme}/templates/fr-FR/catalog/product.tpl**
- **/themes/{mon theme}/templates/catalog/product.tpl**

Gestion des traductions

Nouveau système de traduction

Pour les thèmes, les nouveaux contrôleurs (Symfony) du back-office et les modules développés par PrestaShop.

- Utilisation du système de traduction de Symfony (Composant Translation).
- Regroupement par domaine
- Fichiers au format XLIFF (XML Localisation Interchange File Format)
- Les traductions modifiées par l'utilisateur sont enregistrées en base de données

```
{1 s='Availability date:' d='Shop.Theme.Catalog'}
{1 s='Reference' d='Shop.PDF' pdf='true'}
```

Remarque : L'ancien système de traduction est toujours utilisé par les anciens contrôleurs (legacy) du back-office et les modules qui ne sont pas développés par PrestaShop.

Le domaine Shop

Utilisé pour les traductions du front-office.

Il contient 6 sous domaines :

- Emails : Pour les emails envoyés par la boutique
- PDFs : Pour les PDFs générés par la boutique
- Theme : Les chaînes de caractères provenant du thème (Classique)
- Demo : Pour les pages et les produits de démonstration
- Notifications : Les messages (Warning, Error, Succes) qui peuvent apparaître sur la boutique
- Forms : Les formulaires visibles sur la boutique (Contact, Adresses, Login, etc.)

Le sous domaine Shop.Theme

Contient 4 sous domaines :

- Catalog : pour les pages produits, catégories, etc
- Customer Account : pour la gestion des clients et des commandes
- Checkout : pour la gestion de l'acte d'achat
- Actions : les boutons et les liens qui ne sont pas spécifiques à une page ou un contexte

Le sous domaine Shop.Demo

Contient 2 sous domaines :

- Catalog: pour les produits et catégories (description, titre, etc.)
- Pages: pour les pages CMS (contenu, titre, etc.)

Le sous domaine Shop.Notification

Contient 4 sous domaines :

- Error : pour les messages d'erreurs
- Warning : pour les avertissements
- Success : pour les messages de confirmation
- Info : pour les messages d'informations

Le sous domaine Shop.Forms

Contient 3 sous domaines :

- Labels : Les labels des zones de saisie
- Errors : Les erreurs pour les formulaires
- Help : Les messages d'aide

Le domaine Admin

Utilisé pour les traductions du back-office.

Il existe 12 sous domaines :

- Global : Chaîne générique comme ("Max", "Settings", "Enabled", etc.)
- Actions : Boutons et liens génériques comme ("Save", "Add", "Delete", etc.)
- Notifications : Les messages (Warning, Error, Succes) qui peuvent apparaître sur le back-office
- Navigation : Les textes utilisés par la structure du back-office
- OrdersCustomers : Pour les commandes et les clients

- Catalog : Pour la partie Catalogue du back-office
- Modules : Pour la partie Module du back-office
- Design : Pour la partie Design du back-office
- Shipping : Pour la partie Transport du back-office
- Payment : Pour la partie Paiement du back-office
- International : Pour la partie International du back-office
- Parameters : Pour les paramètres avancés et les paramètres boutique

La gestion des devises

La classe Currency

La classe Currency prend en charge la gestion des devises.

Vous pouvez activer et mettre à jour les taux de conversion depuis le back-office, menu **International > Localisation > Devises**

Depuis la version 1.7, la classe Currency utilise CLDR pour le formatage des montants.

La table Currency permet principalement d'activer une devise et de stocker le taux de conversion.

Lire une devise

La devise courante est disponible dans le Contexte.

```
Context::getContext()->currency;
```

Vous pouvez obtenir une devise à partir d'un id en utilisant la fonction `getCurrencyInstance()`.

Exemple dans la classe Cart :

```
$currency = Currency::getCurrencyInstance((int) $cart->id_currency)
```

Convertir en devise

Pour convertir dans une devise, vous devez utiliser la fonction `Tools::convertPrice`.

```

/**
 * Return price converted.
 *
 * @deprecated since 1.7.4 use convertPriceToCurrency()
 *
 * @param float $price Product price
 * @param object|array $currency Current currency object
 * @param bool $to_currency convert to currency or
 *             from currency to default currency
 * @param Context $context
 *
 * @return float Price
 */
public static function convertPrice(
    $price,
    $currency = null,
    $to_currency = true,
    Context $context = null
)

```

Remarque : La fonction `convertPriceToCurrency` n'existe pas.

Convertir en devise (exemple)

Exemple d'utilisation dans la classe `Cart` :

```

$wrapping_fees = Tools::convertPrice(
    Tools::ps_round(
        $this->getGiftWrappingPrice($withTaxes),
        $computePrecision
    ),
    Currency::getCurrencyInstance((int) $this->id_currency)
);

```

Afficher une devise sur le back-office legacy

Pour afficher un montant dans une devise sur le back-office legacy, vous devez utiliser la fonction `Tools::displayPrice`.

```

/**
 * Return price with currency sign for a given product.
 *
 * @param float $price Product price
 * @param object|array $currency Current currency
 *             (object, id_currency, NULL => context currency)
 *
 * @return string Price correctly formatted (sign, decimal separator...)
 *             if you modify this function, don't forget to modify
 *             the Javascript function formatCurrency (in tools.js)
 */
public static function displayPrice(
    $price,
    $currency = null,
    $no_utf8 = false,
    Context $context = null
)

```

Afficher une devise sur le back-office legacy (exemple)

Exemple d'utilisation dans la classe Cart :

```
return Tools::displayPrice(  
    $cart->getOrderTotal($with_taxes, $type),  
    Currency::getCurrencyInstance((int) $cart->id_currency),  
    false  
);
```

Afficher une devise sur le front-office

Sur PrestaShop 1.7 front-office, la mise en forme est réalisée dans le code PHP avec la classe PriceFormatter.

Exemple dans `/classes/checkout/CheckoutDeliveryStep.php` :

```
$priceFormatter = new PriceFormatter();  
//...  
return $this->getTranslator()->trans(  
    ' (additional cost of %giftcost% %taxlabel%)',  
    array(  
        '%giftcost%' => $priceFormatter->convertAndFormat(  
            $this->getGiftCost()  
        ),  
        '%taxlabel%' => $taxLabel,  
    ),  
    'Shop.Theme.Checkout'  
);
```

Afficher une devise avec Smarty.

Sur PrestaShop 1.6 et sur PrestaShop 1.7 back-office legacy, la mise en forme des informations est réalisée dans Smarty.

`displayPrice` affiche un prix dans la devise courante.

```
{if $order->getTaxCalculationMethod() == $smarty.const.PS_TAX_INC}  
    {displayPrice  
        price=$line.shipping_cost_tax_incl  
        currency=$currency->id  
    }  
{else}  
    {displayPrice  
        price=$line.shipping_cost_tax_excl  
        currency=$currency->id  
    }  
{/if}
```

Afficher une devise avec taxe avec Smarty.

`displayWtPriceWithCurrency` affiche un prix avec taxe et devise.

```

<div class="form-group">
  <label class="col-lg-3 control-label">
    {l
      s='Total spent since registration:'
      d='Admin.Orderscustomers.Feature'
    }
  </label>
  <div class="col-lg-3">
    <p class="form-control-static">
      {displayWtPriceWithCurrency
        price=$customer_stats.total_orders
        currency=$currency
      }
    </p>
  </div>
</div>

```

Remarque : Les fonctions `displayWtPrice` et `convertAndFormatPrice` ne sont plus utilisées dans PrestaShop 1.7.0.

Convertir une devise avec Smarty

Sur PrestaShop 1.6 et sur PrestaShop 1.7 back-office legacy, la mise en forme des informations est réalisée dans Smarty.

`convertPrice` affiche un prix dans une devise.

Exemple d'utilisation :

```

<dt>{l
  s='Total spent since registration'
  d='Admin.Orderscustomers.Feature'
}</dt>
<dd><span class="badge badge-success">
  {displayPrice
    price=Tools::ps_round(
      Tools::convertPrice($customerStats['total_orders'],
        $currency),
      2
    )
    currency=$currency->id
  }
</span></dd>

```

Remarque : La fonction `convertPriceWithCurrency` n'est plus utilisée dans PrestaShop 1.7.0.

Internationalisation des modules

Restrictions par pays

Il est possible de préciser la liste des pays pour lesquelles un module peut être utilisé avec la propriété `limited_countries` de la classe `Module`.

```

/** @var array to store the limited country */
public $limited_countries = array();

```

Exemple pour limiter l'utilisation d'un module à la France et l'Espagne :

```
$this->limited_countries = array('fr','es');
```

L'information est automatiquement copiée dans le fichier cache de configuration des modules (config.xml).

La traduction des modules de PrestaShop

Les modules, qui sont créés par PrestaShop, utilisent la fonction trans du nouveau système de traduction (PrestaShop 1.7.x).

Exemple d'utilisation en PHP :

```
$this->displayName =
    $this->trans(
        'Dashboard Activity',
        array(),
        'Modules.Dashactivity.Admin'
    );
```

Exemple d'utilisation dans un template Smarty :

```
{1
    s='Activity overview'
    d='Modules.Dashactivity.Admin'
}
```

Options possibles :

- "js=1" : indique que cette chaîne peut être contenue dans du code JavaScript.
- "sprintf='any string or number'" : voir les exemples dans la section "Sprintf"

Les traductions sont stockées dans les fichiers

/app/Resources/translations/{iso_code}/{domaine}.{iso_code}.xlf

Exemple pour le domaine Modules.Dashactivity.Admin en Français, les traductions sont stockées dans le fichier **/app/Resources/translations/fr-FR/ModulesDashactivityAdmin.fr-FR.xlf**

Le domaine pour les modules

Les modules développés par PrestaShop utilisent le domaine Modules pour gérer les traductions.

- Modules.{nom_du_module}.Shop : pour la partie front-office du module nom_du_module
- Modules.{nom_du_module}.Admin : pour la partie back-office du module nom_du_module

Des chaînes de caractères "génériques" utilisées par un module peuvent venir directement du domaine Shop ou Admin.

La traduction des modules (non PrestaShop)

Les modules, qui ne sont pas créés par PrestaShop, utilisent la fonction l de l'ancien système de traduction (legacy).

Exemple d'utilisation en PHP :

```
$this->l('Add new root category');
```

Exemple d'utilisation dans un template Smarty :

```
{l s='Add tag' mod='blockcms'}
```

Options possibles :

- "js=1" : indique que cette chaîne peut être contenue dans du code JavaScript.
- "mod='blockcms'" : précise le nom du module
- "sprintf='any string or number'" : voir les exemples dans la section "Sprintf"

Les traductions sont stockées dans les fichiers

/modules/nom_du_module/translations/iso_code.php où iso_code est le code ISO 3166-1 du pays.

Les traductions sont stockées dans un tableau PHP nommé `$_MODULE`, et prenant la forme suivante :

```
$_MODULE['<{blockcms}prestashop>blockmobilecms_dlaa22a3126f04664e0fe3f598994014'] = 'Promotions';
```

La clé d'identification est construite en combinant le nom du module d'où provient la chaîne originale (ex. : "blockcms"), suivi du nom du thème (ex. : "prestashop"), puis du nom du fichier (ex. : "blockmobilecms"), d'un signe souligné, et enfin du hash MD5 de la chaîne elle-même.