

Secteur Tertiaire Informatique  
Filière « Etude et développement »

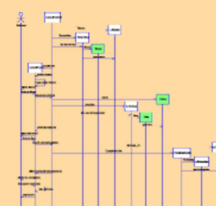
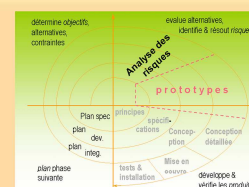
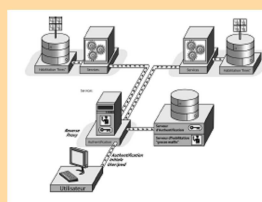
Séquence « Mettre en œuvre une solution  
e-commerce »

Documenter le code PHP

Apprentissage

Mise en pratique

Evaluation





## TABLE DES MATIERES

Table des matières .....	3
1. Pourquoi commenter et documenter du code ? .....	5
2. Que faut-il commenter ? .....	5
3. que faut-il documenter et comment documenter ? .....	7
3.1 Problématique .....	7
3.2 Quels outils pour documenter du code PHP ? .....	10
3.2.1 Doxygen .....	10
3.2.2 phpDocumentor .....	10
3.2.3 phpDox .....	10
3.3 Mettre en œuvre Doxygen pour documenter des modules PrestaShop .....	11

## Objectifs

Ce document a pour objectif de sensibiliser sur l'utilité de bien commenter et documenter le code PHP et d'en générer des pages HTML de documentation. Il présente aussi quelques exemples et outils simples de génération.

## Pré requis

Etre opérationnel dans le développement en langage PHP.

## Outils de développement

Un éditeur de code source, de préférence adapté au langage PHP (NotePad++, codeLobster, Brackets...).

Un outil de génération de documentation compatible avec la version du langage PHP utilisée (Doxygen, PHPDocumentor...).

## Méthodologie

## Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

## Ressources

## Lectures conseillées

## 1. POURQUOI COMMENTER ET DOCUMENTER DU CODE ?

Quand un développeur écrit du code dans un langage de programmation, il doit se préoccuper de la bonne lisibilité de sa production en insérant des instructions particulières de commentaires et documentation. En effet, il existe parfois un écart considérable entre l'intention, le but poursuivi, et sa traduction en suite d'instructions dans le langage de programmation. Tout l'intérêt de la démarche algorithmique est justement de rechercher le meilleur moyen de réaliser le traitement voulu.

Les applications livrées évoluent en permanence et, après création, un programme subit de nombreuses interventions de maintenance (corrective ou évolutive). Comme la mémoire humaine est aussi volatile que celle d'un ordinateur débranché, le développeur se doit de commenter son code afin d'explicitier la logique adoptée, à destination du développeur qui interviendra à sa suite (lui-même ou une autre personne).



**Les commentaires font partie intégrante du code produit par un développeur ; ils ont pour but d'explicitier la logique de programmation adoptée afin de faciliter la maintenance ultérieure du code.**

Quand un développeur réalise une application, il écrit une multitude de portions de code en interrelation mais disséminées dans des fichiers divers ; un fichier source définit une interface qu'un autre utilisera pour définir une classe qui sera instanciée par un contrôleur qui fera appel à une vue elle-même décomposée en diverses parties... Au final, la relecture du code source d'une application (voire d'un simple module) revient à un jeu de piste qui vire parfois au parcours du combattant ! Pour faciliter cette relecture et la réutilisation de composants existants, des systèmes de documentation sont apparus ; ils ont pour but d'explicitier l'architecture des composants et sont destinés à des outils permettant de générer une documentation complète indépendante du code source, en général sous forme de pages Web en interrelation, tout comme la documentation de référence sur un langage ou un framework que le développeur consulte couramment.



**La documentation du code fait partie intégrante des activités du développeur ; elle a pour but de compléter les différents composants écrits en précisant leur structure et leurs relations. La documentation est destinée à être exploitée par un outil permettant de générer des pages d'aide et de documentation technique.**

## 2. QUE FAUT-IL COMMENTER ?

La saisie de commentaires dans du code représente un effort supplémentaire parfois considéré comme inutile car les interpréteurs et compilateurs les ignorent consciencieusement.

La réponse à la question 'que faut-il commenter ?' varie considérablement d'un développeur à un autre, d'une époque à une autre.

Le but étant de faciliter la relecture du code, on peut :

- Commenter chaque instruction ; si cela a été préconisé dans un temps lointain où les langages étaient caractérisés par une syntaxe 'barbare', ce n'est vraiment plus d'actualité !
- Commenter chaque étape de l'algorithme, par exemple 'saisir critère de recherche', 'exécuter recherche en BDD', 'boucle de parcours des clients trouvés'... ; cette pratique courante présente l'avantage que le développeur peut commencer par saisir les commentaires décrivant l'algorithme général, puis y insérer les instructions du langage permettant de réaliser les opérations décrites.

Documenter le code PHP

- Ecrire tout d'abord l'algorithme général du traitement et placer le tout en début de programme, dans le 'cartouche' mémorisant titre et nom du composant, auteurs et dates des interventions.
- Utiliser des noms de variables, de sous-programmes, de classes... significatifs et limiter les commentaires au strict minimum pour expliciter un algorithme particulier (mode de calcul suivant une règle de gestion) ou une syntaxe peu parlante (comme une 'expression régulière') ; cette pratique est de plus en plus couramment utilisée et elle aboutit à des programmes courts mais multiples, presque écrits en langage naturel.
- Pratiquer un mix de tout cela en ne négligeant pas l'importance de ces commentaires pour les opérations de maintenance.

#### Exemple appliqué au langage PHP :

```

<?php
// script controleur lister les films d'un type particulier
// appelé par VCIResa.php
// recherche des films du type demandé en VCIResa.php
// et affichage en tableau avec liens a href personnalisés vers VCIResa3.php
// charger modules externes :
require("presentation/VCIResa2.vue.php"); // module presentation
require("DAO/Video.DAO.PHP"); // module DAO BDD Video
// récupérer film demandé
// pour se connecter a la BDD
$user = "utilweb";
$password = "utilweb";
// récupérer données du type de film voulu
$rowTypeFilm = VideoDAO::RetourneTypeFilm($_GET["t"]);
// recupere liste des films du type voulu
$dataFilms = VideoDAO::ListeFilmsParType($_GET["t"]);
// presenter le résultat
AfficheListeFilms($rowTypeFilm, $dataFilms);
?>

```

Diagramme illustrant la structure du code PHP :

- cartouche** : Encadre les commentaires de documentation au début du script.
- étape** : Encadre la section de chargement des modules externes.
- noms signifiants des variables et méthodes de classe** : Encadre les variables et méthodes utilisées pour récupérer et afficher les données.

### 3. QUE FAUT-IL DOCUMENTER ET COMMENT DOCUMENTER ?

#### 3.1 PROBLEMATIQUE

La documentation explicite la structure d'un composant et ses relations avec d'autres composants.

Dans une programmation orientée objet, il faut documenter a minima :

- chaque classe et interface, globalement, son rôle et ses liens éventuels avec d'autres classes dont les héritages, associations et implémentations d'interfaces ;
- chaque membre de classe ou interface (constante, attribut, propriété, constructeur, méthode, erreurs générées...) en précisant les types de données, les paramètres en entrée et en sortie, les cas d'erreurs, des exemples de code...

Il est bien difficile de préciser ce qui est à documenter car la documentation est destinée à être exploitée par un outil qui générera les pages de documentation ; et chaque outil a ses propres caractéristiques, contraintes et productions ! Alors voici quelques exemples.

Extrait de la documentation de référence du framework Symfony :

Configuration de référence du TwigBundle

Configuration de référence

Configuration du WebProfilerBundle

Configurer dans le noyau (par ex: AppKernel)

Types de formulaire de référence

Fonctions et variables de référence du modèle de formulaire Twig

Fonctions de rendu des formulaires

form(view, variables)

form\_start(view, variables)

form\_end(view, variables)

Index thématique

form\_widget(form.name, variables)

form\_row(form.name, variables)

form\_rest(form, variables)

form\_enctype(form)

Un peu plus sur les « Variables » de formulaire

**form(view, variables)**

Nom de la fonction

Rends le code HTML complet d'un formulaire.

Résumé

```
{# Rends Le formulaire et change La méthode de soumission #}
{{ form(form, {'method': 'GET'}) }}
```

Syntaxe

Vous utiliserez cette fonction pour prototyper ou si vous utilisez des thèmes personnalisés de formulaires. Si vous avez besoin de plus de flexibilité, vous devez utiliser des critères plus bas pour rendre toutes les différentes parties d'un formulaire.

```
{{ form_start(form) }}
    {{ form_errors(form) }}

    {{ form_row(form.name) }}
    {{ form_row(form.dueDate) }}

    <input type="submit" value="Envoyez le formulaire"/>
    {{ form_end(form) }}
```

Exemple

**form\_start(view, variables)**

Rend le début du code HTML d'un formulaire. Cette fonction prends en charge la configuration de la méthode ainsi que la cible de l'action du formulaire. Il inclura bien sur, la propriété `enctype` si le formulaire contient un champs upload.

Source : [http://documentation-symfony.fr/reference/forms/twig\\_reference.html#form-view-variables](http://documentation-symfony.fr/reference/forms/twig_reference.html#form-view-variables)

## Extrait de la documentation de référence du .Net Framework :

...

► IStructuralEquatable Interface

► Queue Classe

► ReadOnlyCollectionBase Classe

▼ SortedList Classe

- SortedList Constructeur
- SortedList Méthodes
- SortedList Propriétés

► Stack Classe

► StructuralComparisons Classe

Index thématique

# SortedList, classe

Nom de la classe

.NET Framework (current version) | Autres versions ▼

Représente une collection de paires clé/valeur triées par les clés et accessible par clé et par index.

**Espace de noms:** System.Collections  
**Assembly:** mscorlib (dans mscorlib.dll)

Résumé

## Hiérarchie d'héritage

Héritage

System.Object  
System.Collections.SortedList

## Syntaxe

Syntaxe

C# C++ F# VB

```
[SerializableAttribute]  
[ComVisibleAttribute(true)]  
public class SortedList : IDictionary, ICollection, IEnumerable,  
    ICloneable
```

## Constructeurs

	Nom	Description
🔗	SortedList()	Initialise une nouvelle instance de la classe SortedList qui est vide, possède la capacité initiale par défaut et est triée suivant l'interface IComparable implémentée par chaque clé ajoutée à l'objet SortedList.
🔗	SortedList(IComparer)	Initialise une nouvelle instance de la classe SortedList qui est vide, possède la capacité initiale par défaut et est triée suivant l'interface IComparer spécifiée.
🔗	SortedList(IComparer, Int32)	Initialise une nouvelle instance de la classe SortedList qui est vide, possède la capacité initiale spécifiée et est triée suivant l'interface IComparer spécifiée.

Liste des membres par catégorie, avec liens vers les pages correspondantes

Source : [https://msdn.microsoft.com/fr-fr/library/system.collections.sortedlist\(v=vs.110\).aspx](https://msdn.microsoft.com/fr-fr/library/system.collections.sortedlist(v=vs.110).aspx)



## Prototype de documentation générée par Doxygen :

Main Page

Classes

Files

Class List

Class Index

Class Hierarchy

Index thématique

D Class Reference

Nom de la classe

#include <diagrams\_d.h>

Inheritance diagram for D:

Diagrammes UML héritage et collaboration

Collaboration diagram for D:

Public Attributes

C m\_c

Additional Inherited Members

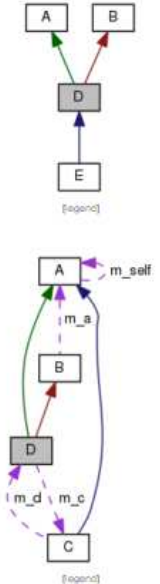
Protected Attributes inherited from A

Member Data Documentation

C D::m\_c

The documentation for this class was generated from the following file:

diagrams\_d.h



Source :

[http://www.stack.nl/~dimitri/doxygen/manual/examples/diagrams/html/class\\_d.html#a9d877c7aa092f423f2a073f3c62fef9c](http://www.stack.nl/~dimitri/doxygen/manual/examples/diagrams/html/class_d.html#a9d877c7aa092f423f2a073f3c62fef9c)

Comme on peut le voir sur ces 3 exemples, les informations générées varient fortement d'un outil à l'autre. En conséquence, les informations de documentation à intégrer dans le code sont spécifiques à l'outil de documentation utilisé, même si la logique générale reste la même.

Dans tous les cas, les données de documentation sont intégrées dans des commentaires particuliers contenant des indicateurs pour le moteur de documentation mais chaque outil impose son propre 'vocabulaire' !



**Les données de documentation sont insérées dans le code source sous forme de commentaires spécifiques à l'outil de génération de documentation utilisé.**

Documenter le code PHP

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

## 3.2 QUELS OUTILS POUR DOCUMENTER DU CODE PHP ?

Dans le monde Open Source dont fait partie le langage PHP, les outils de génération de documentation sont nombreux et gratuits. Avantage : on a le choix. Inconvénient : ces outils sont en général mal stabilisés, mal documentés (ce qui est un comble !), peinent à suivre l'évolution rapide des technologies et sont parfois difficiles à installer et paramétrer (manipulations en ligne de commande...).

Actuellement, 2 outils principaux sont relativement fiables et utilisés.

### 3.2.1 Doxygen

Doxygen est un générateur de documentation destiné à différents langages, dont le PHP, ce qui est intéressant quand on travaille en équipes et avec différents langages selon les projets. Son vocabulaire est très complet et il permet de générer des graphes intégrés aux pages de textes comme l'a montré l'exemple plus haut.

Pour en savoir plus : <https://fr.wikipedia.org/wiki/Doxygen>

La documentation de référence se trouve sur : [www.doxygen.org](http://www.doxygen.org)

Et il existe un petit tutoriel bien pratique en Français :  
<http://axiomcafe.fr/tutoriel-documenter-un-code-avec-doxygen>

### 3.2.2 phpDocumentor

phpDocumentor est un outil destiné à générer de la documentation de code PHP ; il est lui-même écrit en langage PHP. Son installation n'est pas toujours aisée et peut poser des problèmes de compatibilité avec la version du moteur PHP déjà installé... Cependant, il est fréquemment utilisé.

Pour en savoir plus : <http://phpdoc.org/>

### 3.2.3 phpDox

phpDox adopte une démarche originale en ce sens que l'outil ne génère pas des pages HTML mais des documents XML bien structurés ; il 'suffit' donc de leur associer une feuille de styles CSS ou une transformation XSL pour générer des pages Web de documentation.

Pour en savoir plus : <http://phpdox.de/>

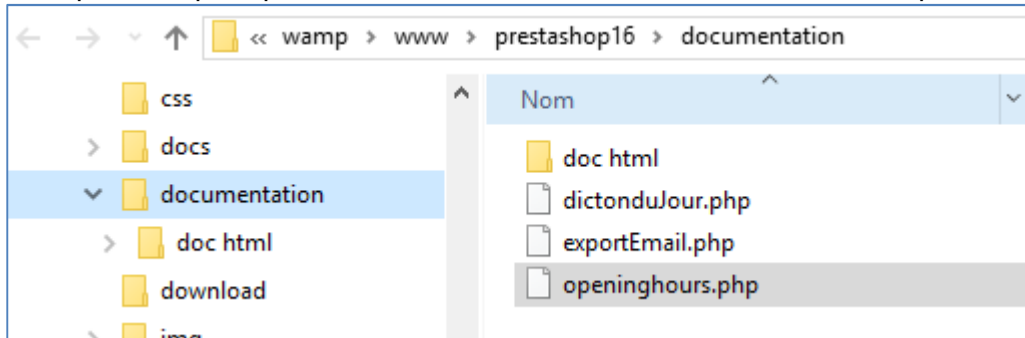
### 3.3 METTRE EN ŒUVRE DOXYGEN POUR DOCUMENTER DES MODULES PRESTASHOP

Il s'agit tout d'abord de télécharger et d'installer le logiciel. Dès lors on dispose d'une application Doxywizard disponible dans le menu principal Windows ; c'est là que ça se passe pour générer la documentation.

Mais avant de lancer cette application, un peu d'organisation :

- Il est utile de créer un dossier pour les composants de code à documenter et un autre pour la documentation elle-même. L'idéal resterait bien entendu de ne pas dupliquer les fichiers de code PHP mais par défaut Doxygen ne peut scanner plusieurs dossiers à la fois. On recopiera donc ponctuellement les composants PHP à documenter dès lors qu'ils seront 'finis-finis'.

Pourquoi ne pas placer tout cela dans le dossier de la boutique PrestaShop ?



- Et il est bien sûr nécessaire d'enrichir nos modules de commentaires avec les marques spéciales à destination de Doxygen. Cette opération sera effectuée sur les fichiers PHP de module dans leurs dossiers d'origine, avant recopie dans le dossier de documentation.

En parcourant les aides et tutoriaux mentionnés plus haut, on retient que les données essentielles à ajouter sont :

- Les marques de commentaires spéciaux pour Doxygen, par exemple dans leur forme `/** */` au niveau des classes et des membres à documenter
- Dans ces marques, la définition des attributs :
  - `\brief \author \version \date` au niveau de la classe
  - `\fn \brief \details \return` et `\param` pour chacune des méthodes

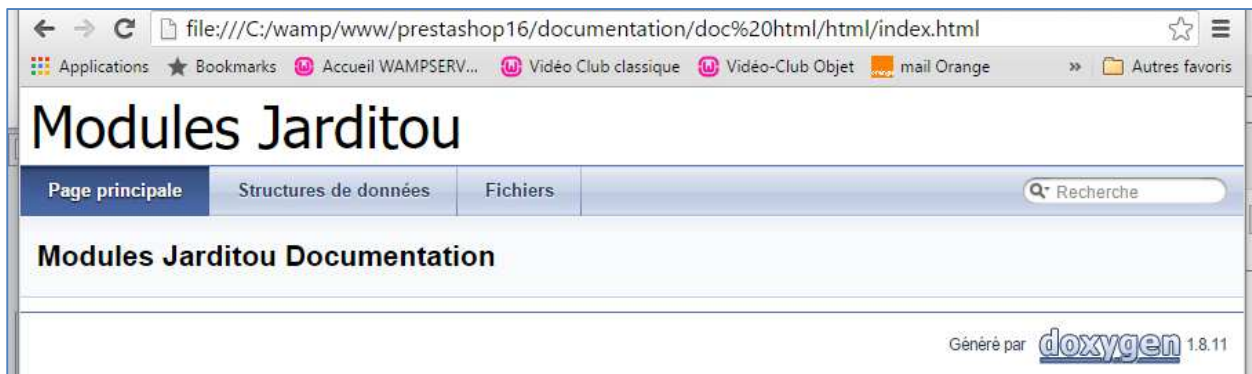
Cela suffit car nos classes de modules pour PrestaShop restent très succinctes puisqu'elles dérivent de la classe de base `Module` déjà tellement riche !

On peut alors lancer l'assistant Doxywizard et se laisser guider en 4 étapes de paramétrage général, puis lancer la génération (bouton `Run Doxygen`) et consulter immédiatement le résultat dans un navigateur (bouton `Show HTML output`). C'est presque magique !

Pour un résultat optimal, il sera quand même nécessaire de régler quelques paramètres dans l'onglet `Expert` comme la langue utilisée pour la rédaction, la prise en compte des éventuels membres privés ou des options de présentation. En passant, on peut remarquer l'option permettant de donner une liste de dossiers à scanner, ce qui permet d'éviter la copie inutile et dangereuse de nos scripts PHP...

Documenter le code PHP

Voici quelques extraits de la documentation concernant 3 modules Prestashop générée à peu de frais :



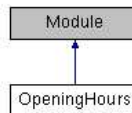
# Modules Jarditou

[Page principale](#)[Structures de données](#)[Fichiers](#)[Structures de données](#)[Index des structures de données](#)[Hiérarchie des classes](#)[Champs de donnée](#)[Fonctions membres publiques](#) | [Fonctions membres protégées](#)

## Référence de la classe OpeningHours

Module affichage horaires d'ouverture. [Plus de détails...](#)

Graphe d'héritage de OpeningHours:



### Fonctions membres publiques

#### **\_\_construct ()**

Constructeur sans parametres. [Plus de détails...](#)

#### **install ()**

Installe le module. [Plus de détails...](#)

#### **uninstall ()**

Desinstalle le module. [Plus de détails...](#)

#### **getContent ()**

Configuration du module. [Plus de détails...](#)

## Description détaillée

Module affichage horaires d'ouverture.

#### **Auteur**

Benoit Hezard

#### **Version**

1.0

#### **Date**

29/02/2016

Définition à la ligne 10 du fichier **openinghours.php**.

## Documentation des constructeurs et destructeur

#### **\_\_construct ( )**

Constructeur sans parametres.

Initialise les membres herites name, tab, version, author, need\_instance, ps\_versions\_compliancy, bootstrap  
Definit displayName, description, confirmUninstall

Définition à la ligne 18 du fichier **openinghours.php**.

Documenter le code PHP

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

**Methode install ( )**

Installe le module.

Installe le module et associe les Hook leftColumn, rightColumn et header

Définition à la ligne **42** du fichier **dictonduJour.php**.

**Methode uninstall ( )**

Desinstalle le module.

Desinstalle le module

Définition à la ligne **58** du fichier **dictonduJour.php**.

La documentation de cette classe a été générée à partir du fichier suivant :

- C:/wamp/www/prestashop16/modules/dictondujour/**dictonduJour.php**

Généré par **doxygen** 1.8.11

## **CREDITS**

### **ŒUVRE COLLECTIVE DE l'AFPA**

**Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services**

#### **Equipe de conception (IF, formateur, mediatiseur)**

B. Hézard - Formateur

Ch. Perrachon – Ingénieure de formation

**Date de mise à jour : 29/02/16**

### **Reproduction interdite**

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Documenter le code PHP

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »