PhpDocumentor : l'usine à documentation pour PHP

1ère partie

Commenter des sources est une tâche ingrate, que tout développeur devrait pourtant faire consciencieusement. Cela pourrait lui servir un jour pour comprendre son propre code ! PhpDocumentor est un générateur de documentation pour code PHP.

Lorsque l'on démarre un projet, il est habituel de parfaitement commenter le code. Chaque classe, chaque fonction, chaque variable se voit décrite soigneusement, et tout va pour le mieux... jusqu'au moment ou le temps presse, qu'il faut aller plus vite. On a alors tendance à négliger les commentaires. C'est là ce que l'on appelle " l'entropie du code ". Mais le jour où il faut modifier le source, quelques mois plus tard, on regrette souvent le manque de documentation... Ceci est une bonne raison pour documenter convenablement le code.
phpDocumentor permet en effet de générer une documentation technique à partir des sources d'une application en PHP. Ou plus exactement, à partir des commentaires que le développeur y aura placé. Ce principe, exploité depuis longtemps dans d'autres langages, faisait défaut à PHP. Et même si phpDocumentor n'est pas tout récent, il aura fallu attendre ses dernières versions pour pouvoir l'employer en production sur des projets d'importance. Pour qu'il puisse exploiter vos commentaires, il faut les formater d'une certaine manière, afin qu'ils puissent être " parsés " convenablement. Il vous faudra insérer tous vos commentaires dans des " DocBlocks ". Leur syntaxe est très simple :

```
/**
* Ceci est un bloc de commentaire
*
* Il permet de décrire le code qui le suit
* immédiatement dans le source
*/
```

Le principe est simple : un DocBlock commence par la séquence " /** " et se termine par " */ ". Chaque ligne entre ces deux séquences doit commencer par une étoile (*), sans quoi elle sera ignorée lors de l'analyse. La phrase " Ceci est un bloc de commentaire " sera considérée comme la description courte de l'élément commenté, alors que les phrases suivantes constitueront la description longue. Celle-ci se termine soit par la fin du bloc, soit par une liste de " tags ". Ces tags sont interprétés de manière spécifique pour améliorer l'analyse du code, et de fait, la mise en page des documentations. La description longue est facultative : vous pouvez ne mettre que la description courte suivie des tags (nous reviendrons sur les tags plus loin dans cet article), ou même vous contenter de celle-ci.
Notez qu'il est précisé que le bloc de commentaire est rattaché au code qui suit immédiatement la fin du bloc. Voici les éléments qui peuvent être commentés par un bloc autonome : les déclaration de constantes (define()), les inclusions de fichiers (include() ou require()), les variables globales, les classes, les propriétés de classes et les fonctions. A cette règle une seule exception : les blocs à échelle du fichier. Ces blocs spécifiques permettent de décrire le contenu entier d'un fichier. Un bloc de commentaire est considéré comme décrivant le fichier, s'il est le premier bloc du fichier et qu'il est suivi d'un autre bloc de commentaire, ou bien si il contient le tag @package.
Exemple :

```
<?
/**
* Bloc de fichier
*
* @package MonApplication
*/

class MaClasse{

}
?>
```

Dans cette situation, phpDocumentor associera le bloc au fichier et non pas à la classe qui suit immédiatement le bloc. Si le tag @package n'est pas présent et que vous n'écrivez pas de bloc de commentaire pour "MaClasse" (ce qui est mal, ne l'oublions pas !), alors le premier bloc sera associé quand même à la classe.
Les tags permettent de préciser la nature de l'information qui le suit directement. Il existe de nombreux tags, mais soyez rassuré, il suffit d'en maîtriser seulement quelques-uns pour documenter parfaitement ses sources. Les autres permettent d'aller encore plus loin, mais restent facultatifs dans la plupart des cas.
Nous venons donc de voir le tag @package. Ce tag permet de diviser les éléments d'une application en différents modules, de manière à scinder la documentation en plusieurs parties pour la rendre plus lisible et plus cohérente. Ce tag est optionnel, et est utilisé notamment dans les blocs à l'échelle du fichier, car si celui-ci est défini, tous les éléments du fichier seront rattachés automatiquement au même " package ". Si vous l'omettez, vous pouvez définir un module par défaut au moment de la génération de la doc, qui sera donc utilisé par tous les blocs pour lesquels le package n'est spécifié, ni explicitement (tag @package), ni implicitement (héritage du tag @package du fichier). Dans un bloc de fichier, les tags les plus souvent utilisés sont les suivants :

```
@package, que nous venons de voir
@author qui permet de préciser l'auteur du code
@copyright pour ... le copyright
```

@version, pour la version évidemment.

Un bloc complet peut donner ça :

```
/**
* Fonctions diverses
*
* Librairies de fonctions auxiliaires
* pour MonApplication
*
* @author Gauthier Delamarre
* @copyright 2003
* @package MonApplication
* @version 1.2
*/
```

Dans un bloc de fonction ou de classe, il est en général inutile de répéter le nom de l'auteur et le copyright, en revanche, les tags @packages et @version sont fréquemment utilisés.
Les choses deviennent plus intéressantes lorsqu'il s'agit de commenter une déclaration de fonction. Car là, il faudra prévoir un commentaire par paramètre de la fonction. Ceux-ci sont commentés directement dans le bloc de la fonction, au moyen de tags @param. Il doit y en avoir autant que de paramètres, et ils doivent être listés dans le même ordre que dans la déclaration de la fonction. Le tag @param attend deux chaînes de caractères : la première pour déterminer le type de l'argument attendu, la seconde pour sa description. Vous pouvez le cas échéant utiliser un tag @return pour indiquer la nature de la valeur retournée par la fonction.

```
/**
* Génération d'un triplet hexadécimal
*
* Retourne une chaîne de 6 caractères à partir de
* trois valeurs décimales comprises entre 0 et 255
*
* @param int Valeur décimale pour le rouge
* @param int Valeur décimale pour le vert
* @param int Valeur décimale pour le bleu
*
* @return string
*/
function htmlcolor($red,$green,$blue) {
...
}
```

Le type des arguments est une chaîne libre, pour la simple raison qu'il peut s'agir d'un type personnalisé. En PHP, ce cas se produit lorsqu'une fonction attend en paramètre une instance de classe. Dans ce cas, phpDocumentor fera automatiquement un lien vers la déclaration de la classe citée dans le champ " type ". Si vous affectez des valeurs par défaut aux arguments, celles-ci seront prises en compte automatiquement, il n'est donc pas nécessaire de les remettre dans la description de l'argument correspondant.
Note : la syntaxe du bloc de commentaire est la même si vous déclarez une fonction isolée ou si elle est une méthode d'une classe.

Parmi les autres éléments qu'il est extrêmement important de commenter, il ne faut pas oublier les propriétés des classes. Bien qu'il ne soit pas obligatoire de déclarer les membres d'une classe en PHP (pour le moins avec la configuration par défaut), il est préférable de les lister avec le mot clef " var " au début de la déclaration de la classe. Dans ce cas, vous pouvez commenter chaque variable avec un bloc entier. La syntaxe de ces blocs est la même que pour les autres éléments, au détail près qu'il doivent contenir le tag @var, suivi du seul type de la variable (sa description étant fournie en en-tête du bloc, dans sa version courte, mais aussi si vous le souhaitez dans sa version longue).

```
/**
* Déclaration de MaClasse
*/
class MaClasse {

  /**
   * Age du capitaine
   *
   * @var int
   */
  var $ageDuCapitaine;

  /**
   * Nom du capitaine
   *
   * @var string
   * @access protected
   */
  var $nomDuCapitaine;
}
```

Dans le deuxième exemple, nous avons ajouté un tag @access. Ce tag peut prendre deux valeurs : protected ou private. Dans le premier cas, cela permet d'informer l'utilisateur que la variable ne doit pas être modifiée directement, mais seulement par la classe elle-même ou ses héritières. Cette notion n'existe pas (pour l'instant) au niveau de l'interpréteur PHP, mais il peut être nécessaire de la respecter quand même. Si vous spécifiez une variable comme étant " private ", elle ne sera pas mentionnée dans la documentation (à moins d'activer l'option " parseprivate " de phpDocumentor).
Il ne vous reste plus qu'à tester le système en téléchargeant phpDocumentor sur phpdoc.org. Une très bonne interface web est fournie avec le package, rendant l'utilisation du logiciel vraiment triviale. Le plus simple étant d'utiliser un fichier .ini, que vous pouvez créer en copiant le fichier default.ini et en l'adaptant à votre projet, puis que vous placez dans le sous répertoire " user ". Inutile de préciser que le fichier default.ini est abondamment commenté. N'oubliez pas de faire votre choix parmi les nombreux thèmes proposés, sans quoi la génération de votre documentation risque de prendre ... un temps certain. Il vous suffit ensuite, depuis l'interface web, de sélectionner votre fichier .ini et de cliquer sur " Go ".
Nous verrons dans la suite comment utiliser certaines fonctionnalités avancées de cet outil véritablement indispensable.

Gauthier Delamarre