**Le PHPDoc dans le code source**

Le code source de SPIP utilise le format PHPDoc pour documenter ses fonctions et ses fichiers. Nous décrivons ici les usages.

**Présentation**

Les « docblocks » sont des commentaires PHP spécifiques dans le code pour documenter celui-ci. Ils commencent par /\*\* et se terminent par \*/. En général, le signe \* crée un filet vertical :

*/\*\**

*\* Ceci est un docblock*

*\*/*

Par convention, la première ligne sert de description courte de la fonction. Elle peut être suffisante ou non. Si une description plus longue doit être apportée, il faut sauter une ligne entre les deux :

*/\*\**

*\* Ceci est la description courte, généralement sur une seule ligne*

*\**

*\* Ceci est une description longue, facultative. Elle commence après la ligne*

*\* vide qui suit la description courte*

*\*/*

Le contenu à l’intérieur de ces blocs s’écrit en utilisant une syntaxe [Markdown](http://michelf.ca/projets/php-markdown/syntaxe/). Celle-ci bien que très pratique a quelques subtilités. La plus importante à connaître, c’est que toute liste d’éléments a aussi une ligne vide avant :

*/\*\**

*\* Ceci est la description courte, généralement sur une seule ligne*

*\**

*\* Ceci est une description longue*

*\**

*\* - Et ceci est un élément de liste (il doit y avoir une ligne vide avant)*

*\* - Et un autre élément.*

*\*/*

**Insérer du code**

On peut citer du code de trois manières, entre `, indenté de 4 espaces, ou entre multiples ``` (ou un mélange des 2 derniers) :

*/\*\**

*\* Ceci est `du code en ligne`*

*\**

*\* Ceci est une description longue*

*\**

*\*     Ceci est aussi un bloc de code sur plusieurs lignes.*

*\*     Il faut une ligne vide et une indentation d'au moins 4 espaces.*

*\**

*\* ```*

*\* Ceci est aussi du code sur plusieurs lignes*

*\* ```*

*\*      ```*

*\*      Ceci est encore du code*

*\*      ```*

*\*/*

**Les tags**

Les ’tags’ sont comme des variables qui servent à décrire notre fonction. Ils donnent des informations en plus par rapport aux descriptions courtes et longues. Un tag s’écrit `@nom\_du\_tag` et il peut s’en suivre un certain contenu, en fonction du tag.

Voici donc quelques exemples de tags :

*/\*\**

*\* Description courte*

*\**

*\* @api*

*\* @see chemin()*

*\* @link http://www.spip.net/...*

*\*/*

**Liste des tags de PHPDoc**

Sur la catégorisation (généralement pour l’entête de fichier) :

* @package x : Indique que l’élément regroupé dans x
* @package SPIP\Core\x : Indique que l’élément est du SPIP, dans le Core, puis dans un sous élément x
* @package SPIP\Breves\x : Indique que l’élément regroupé dans le plugin Brèves, puis dans le sous élément x.

**La liste des sous éléments est encore à définir plus précisément.** On trouve actuellement : Installation, Autorisations, Pipelines, Action, de façon assez générique. Ceci reste à préciser.

Sur l’usage de l’élément :

* @api : Indique que l’élément est une API
* @deprecated[ texte] : Indique que l’élément est déprécié est ne doit plus être utilisé. Le texte peut servir à indiquer l’alternative à utiliser.

Sur les liens entre éléments et documentations :

* @see element [texte] : indique un renvoie vers un autre élément
* @uses element [texte] : indique que la fonction utilise cet élément
* @link url [texte] : indique un renvoie vers une ressource externe

Sur les paramètres :

* @param type $var [texte] : indique que le paramètre $var de la fonction est du type indiqué. On peut indiquer plusieurs possibilités avec | : int|string|null $var. Lorsqu’il y a plusieurs paramètres, et donc plusieurs fois le tag `@param`, veillez à conserver le même ordre que les paramètres de la fonction, et les mêmes noms de variable.
* @return type : pareil que @param pour les retours de la fonction. Notez que si la fonction ne retourne rien (aucun mot clé return dans le code de la fonction), vous pouvez omettre @return, ou indiquer @return void. Notez aussi que @return null est différent de @return void. Vous pouvez cumuler plusieurs retours bien évidemment tel que @return void|int|string

Autres tags :

* @example [texte] sert à donner un ou plusieurs exemples. S’il y a du code dans l’exemple, il doit être échappé selon l’usage, par exemple être mis entre ```.
* @todo [texte] indique quelque chose qui reste à faire !

**Tags spécifiques à SPIP**

* @note [texte] Indique un commentaire plutôt en relation avec le code interne de la fonction
* @balise Indique que la fonction compile une balise #XX
* @filtre indique que la fonction peut servir de filtre dans un squelette
* @critere indique que la fonction compile un critère
* @boucle indique que la fonction compile une boucle
* @pipeline xx indique que la fonction est traversée lors de l’appel par le pipeline xx. La fonction peut donc compléter des données sur le pipeline.
* @pipeline\_appel xx indique que la fonction appelle la fonction pipeline() avec le pipeline xx pour en obtenir des données

**Exemples pour des fonctions**

*/\*\**

*\* Donne le nom de la copie locale de la source*

*\**

*\* Soit obtient l'extension du fichier directement de l'URL de la source,*

*\* soit tente de le calculer.*

*\**

*\* @uses nom\_fichier\_copie\_locale()*

*\* @uses recuperer\_infos\_distantes()*

*\**

*\* @param string $source*

*\*      URL de la source distante*

*\* @return string*

*\*      Nom du fichier calculé*

*\*\*/*

**function** fichier\_copie\_locale($source){

*/\*\**

*\* Retourne le nombre de lignes d’une ressource de sélection obtenue*

*\* avec `sql\_select()`*

*\**

*\* @api*

*\* @see sql\_select()*

*\* @see sql\_countsel()*

*\**

*\* @param Ressource $res*

*\*     Ressource SQL*

*\* @param string $serveur*

*\*     Nom du connecteur*

*\* @param bool|string $option*

*\*     Peut avoir 2 valeurs :*

*\**

*\*     - true -> executer la requete*

*\*     - continue -> ne pas echouer en cas de serveur sql indisponible*

*\* @return bool|string*

*\*     - int Nombre de lignes,*

*\*     - false en cas d'erreur.*

*\*\*/*

**function** sql\_count($res, $serveur='', $option=**true**) {

**En-tête de fichiers**

PHPDocumentor n’est pas content lorsqu’il n’a pas de phpdoc pour un en-tête de fichier. Comme pour les fonctions, il est bon que chaque fichier décrive plus ou moins laconiquement leur contenu. Exemples :

(mauvais exemple, il manque les accents !)

*/\*\**

*\* Definition de l'API SQL*

*\**

*\* Ce fichier definit la couche d'abstraction entre SPIP et ses serveurs SQL.*

*\* Cette version 1 est un ensemble de fonctions ecrites rapidement*

*\* pour generaliser le code strictement MySQL de SPIP <= 1.9.2*

*\* Des retouches sont a prevoir apres l'experience des premiers portages.*

*\* Les symboles sql\_\* (constantes et nom de fonctions) sont reserves*

*\* a cette interface, sans quoi le gestionnaire de version dysfonctionnera.*

*\**

*\* @package SPIP\Core\SQL\API*

*\* @version 1*

*\*/*

Ou plus simplement :

*/\*\**

*\* Gestion de l'action activer\_plugins*

*\**

*\* @package SPIP\Core\Plugins*

*\*\*/*

**Éléments obligatoires et optionnels**

PHPDocumentor loge une erreur dès que certains PHPDoc sont absents. Ils sont normalement obligatoires sur ces éléments :

* entête de fichier
* fonction
* classe
* propriété de classe
* méthode de classe

Il est aussi possible d’en mettre sur les constantes déclarées (mais alors juste avant sa déclaration, pas avant un éventuel if qui la précède. L’autodoc peut alors générer la liste des constantes présente dans le code source, avec une belle description à côté.

Exemple :

if (![defined](http://www.php.net/defined)('\_DIR\_RESTREINT\_ABS')) {

*/\*\* le nom du repertoire ecrire/ \*/*

[define](http://www.php.net/define)('\_DIR\_RESTREINT\_ABS', 'ecrire/');

}