Les ‘Docblocks’

Les blocs de documentation, ou **docblocks**, sont des commentaires spéciaux utilisés pour documenter le code. Ils sont couramment utilisés avec des générateurs de documentation comme PHPDoc. Les docblocks aident non seulement à générer de la documentation, mais aussi à fournir des informations utiles aux développeurs et aux IDE pour des fonctionnalités comme l’auto-complétion et les analyses statiques.

PhpUserFilter.php

```
22  /**
23   * @link https://php.net/manual/en/php-user-filter.filter.php
24   * @param resource $in <p> is a resource pointing to a <i>bucket brigade</i< which
25   * contains one or more <i>bucket</i> objects containing data to be filtered.</p>
26   * @param resource $out <p>is a resource pointing to a second bucket brigade into
27   * which your modified buckets should be placed.</p>
28   * @param int &$consumed <p>which must <i>always</i> be declared by reference,
29   * should be incremented by the length of the data which your filter reads in
30   * and alters. In most cases this means you will increment consumed by
31   * <i>$bucket->datalen</i> for each <i>$bucket</i>.</p>
32   * @param bool $closing <p>If the stream is in the process of closing (and therefore
33   * this is the last pass through the filterchain), the closing parameter will be set
34   * to <b>TRUE</b>
35   * @return int <p>
36   * The <b>filter()</b> method must return one of
37   * three values upon completion.
38   * </p><table>
39   *
40   * <thead>
41   * <tr>
42   * <th>Return Value</th>
43   * <th>Meaning</th>
44   * </tr>
45   *
46   * </thead>
47   *
48   * <tbody class="tbody">
49   * <tr>
50   * <td><b>PSFS_PASS_ON</b></td>
51   * <td>
52   * Filter processed successfully with data available in the
53   * <code class="parameter">out</code> <em>bucket brigade</em>.
54   * </td>
55   * </tr>
56   *
57   * <tr>
58   * <td><b>PSFS_FEED_ME</b></td>
59   * <td>
60   * Filter processed successfully, however no data was available to
61   * return. More data is required from the stream or prior filter.
62   * </td>
63   * </tr>
```

Voici un guide sur la façon d’écrire des docblocks pour différentes parties de votre code PHP.

Docblock pour une Classe

```php
/**
 * Class Car
 *
 * This class represents a car.
 *
 * @package Vehicle
 */
class Car {
    // Code de la classe...
}
```

```
1
2  /*\*
3   \* La classe Bidule permet de gérer le bidulage d'un projet.
4   \*
5   \* @author Toto <toto[@]toto.com>
6   \*/
7  class Bidule
8  {
9  ⋯
10 }
```

Docblock pour une Propriété

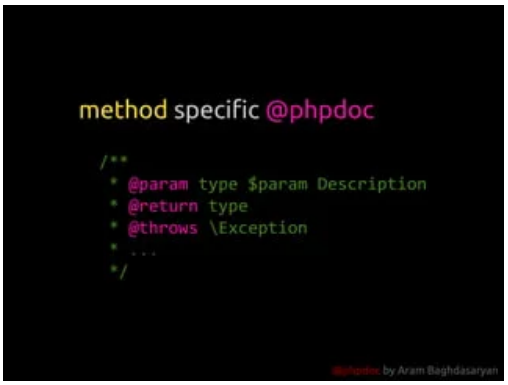
```php
/**
 * @var string The make of the car.
 */
private $make;
```

```
 * @property string $No
 * @property string $Description
 * @property string $Description 2
 * @property string $Shelf Front Edge 1
 * @property string $Shelf Front Edge 2
```

Docblock pour une Méthode

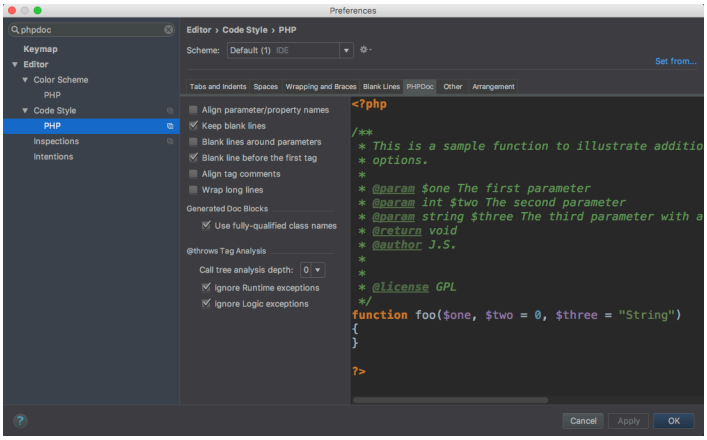
```php
/**
 * Car constructor.
 *
 * Initializes a new instance of the Car class.
 *
```

```
 * @param string $make The make of the car.
 * @param string $model The model of the car.
 */
public function __construct(string $make, string $model) {
    $this->make = $make;
    $this->model = $model;
}
```



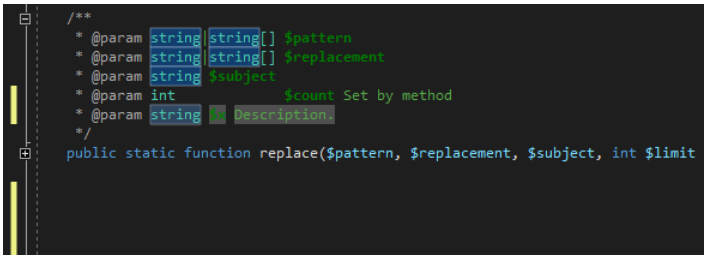
Docblock pour une Méthode avec Retour

```
/**
 * Get the make of the car.
 *
 * @return string The make of the car.
 */
public function getMake(): string {
    return $this->make;
}
```



Docblock pour une Méthode avec Paramètres

```
/**
 * Set the make of the car.
 *
 * @param string $make The make of the car.
 * @return void
 */
public function setMake(string $make): void {
    $this->make = $make;
}
```



Docblock pour une Méthode avec Exceptions

```
/**
 * Start the car.
 *
 * This method starts the car and throws an exception if the car cannot start.
 *
```

```php
 * @throws Exception If the car cannot start.
 * @return void
 */
public function start(): void {
    if (!$this->canStart()) {
        throw new Exception("The car cannot start.");
    }
    // Démarre la voiture...
}
```

```php
/**
 * Charge the customer card
 *
 * @throws \Tc24StripeBundle\Exceptions\ConnectionException
 * @throws \Tc24StripeBundle\Exceptions\PaymentException
 * @throws \Tc24StripeBundle\Exceptions\InvalidRequest
 *
 * @return \Stripe\ApiResource
 */
protected function executeCharge()
{
    try {
        return Charge::create($this->chargeArray, $this->chargeOptions);
    }
    catch (Card $e) {
        $body = $e->getJsonBody();
        throw new PaymentException($body['error']['message']);
    }
    catch (ApiConnection $e) {
        throw new ConnectionException($e->getMessage());
    }
    catch (InvalidRequest $e) {
        throw new \Tc24StripeBundle\Exceptions\InvalidRequest($e->getMessage());
    }
}
```

Exemple Complet

Voici un exemple complet d'une classe avec des docblocks appropriés :

```php
/**
 * Class Car
 *
 * This class represents a car.
 *
 * @package Vehicle
 */
class Car {
    /**
     * @var string The make of the car.
     */
    private $make;
```

```php
/**
 * @var string The model of the car.
 */
private $model;

/**
 * Car constructor.
 *
 * Initializes a new instance of the Car class.
 *
 * @param string $make The make of the car.
 * @param string $model The model of the car.
 */
public function __construct(string $make, string $model) {
    $this->make = $make;
    $this->model = $model;
}

/**
 * Get the make of the car.
 *
 * @return string The make of the car.
 */
public function getMake(): string {
    return $this->make;
}

/**
 * Set the make of the car.
 *
 * @param string $make The make of the car.
 * @return void
 */
public function setMake(string $make): void {
    $this->make = $make;
}

/**
```

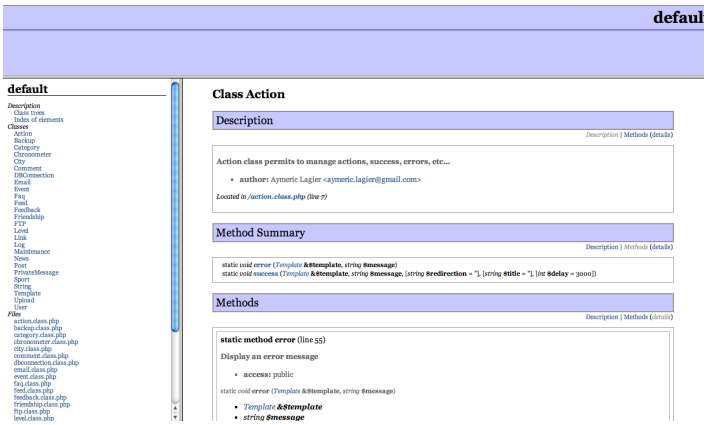
```
 * Get the model of the car.
 *
 * @return string The model of the car.
 */
public function getModel(): string {
    return $this->model;
}

/**
 * Set the model of the car.
 *
 * @param string $model The model of the car.
 * @return void
 */
public function setModel(string $model): void {
    $this->model = $model;
}

/**
 * Start the car.
 *
 * This method starts the car and throws an exception if the car cannot star
 *
 * @throws Exception If the car cannot start.
 * @return void
 */
public function start(): void {
    if (!$this->canStart()) {
        throw new Exception("The car cannot start.");
    }
    // Code pour démarrer la voiture...
}

/**
 * Check if the car can start.
 *
 * @return bool True if the car can start, false otherwise.
 */
```

```php
    private function canStart(): bool {
        // Logique pour vérifier si la voiture peut démarrer...
        return true;
    }
}
```



Principales Balises Docblock

Voici quelques balises courantes utilisées dans les docblocks :

- `@param` : Décrit un paramètre de méthode ou de fonction.
- `@return` : Décrit la valeur de retour d'une méthode ou d'une fonction.
- `@var` : Décrit une variable ou une propriété.
- `@throws` : Décrit les exceptions que peut lancer une méthode ou une fonction.
- `@package` : Indique le package auquel appartient la classe.
- `@see` : Référence un autre élément de la documentation.
- `@deprecated` : Indique que l'élément est obsolète.

phpDoc Tags

Tag	Usage	Description
@name	global variable name	Specifies an alias for a variable. For example, $GLOBALS['myvariable'] becomes $myvariable
@magic		phpdoc.de compatibility "phpDocumentor tags".
@package	name of a package	Documents a group of related classes and functions.
@param	type [$varname] description	
@return	type description	This tag should not be used for constructors or methods defined with a *void* return type.
@see		Documents an association to another method or class.
@since	version	Documents when a method was added to a class.
@static		Documents a static class or method
@staticvar		Documents a static variable's use in a function or class
@subpackage		
@throws		Documents an exception thrown by a method.
@todo		Documents things that need to be done to the code at a later date.
@var	type	a data type for a class variable
@version		Provides the version number of a class or method.

http://manual.phpdoc.org/HTMLSmartyConverter/PHP/phpDocumentor/tutorial_tags.pkg.html

Conclusion

En utilisant ces docblocks, vous pouvez rendre votre code PHP beaucoup plus facile à comprendre et à maintenir, tout en facilitant la génération automatique de documentation avec des outils comme PHPDoc.