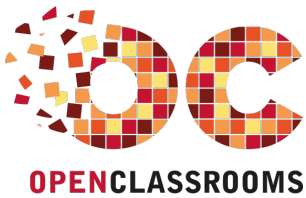
Les espaces de noms en PHP

Par Victor Thuillier (vyk12)



www.openclassrooms.com

Dernière mise à jour le 30/10/2010

Sommaire

Les espaces de noms en PHP

Par Victor Thuillier (vyk12)

Mise à jour : 30/10/2010
Difficulté : Intermédiaire

Bonjour à tous.

Je vais ici vous présenter un côté de PHP bien intéressant : il s'agit des **espaces de noms** (ou **namespaces**, c'est d'ailleurs comme ça que je vais les désigner pour la suite du tutoriel). Vous découvrirez comment séparer vos constantes, fonctions et classes dans différents espaces afin d'éviter tout conflit : vous pourrez ainsi avoir plusieurs constantes, fonctions ou classes du même nom dans la même application !

Les **namespaces** sont disponibles depuis PHP 5.3. Veillez donc bien à avoir cette version, sinon vous aurez une belle erreur de parsage.

Sommaire du tutoriel :

Présentation des namespaces

Découvrons ce que sont les namespaces. L'implémentation d'un tel module dans un langage comme PHP ne se fait pas sur un coup de tête.

Le problème

Ce module a été créé afin de répondre à une problématique toute simple : on ne peut pas créer deux constantes, deux fonctions ou deux classes portant le même nom. En effet, si vous réalisez une telle chose, vous aurez une belle erreur vous indiquant que la constante, la fonction ou la classe a déjà été déclarée. Il a donc fallu trouver un système qui puisse être plus souple et résoudre le problème.

La solution

Nos chers développeurs n'ont pas eu à réfléchir longtemps : d'autres langages (comme le C++) utilisaient déjà un système qui permet ce genre de choses. Ils l'ont donc repris et adapté au langage. Le principe est tout simple, et vous le connaissez déjà plus ou moins : l'idée est la même que l'organisation des fichiers.

On va s'imaginer votre code PHP représenté sous forme de fichiers. Chaque constante, fonction et classe représente un fichier. Actuellement, ils sont tous placés à la racine. Résultat : si deux constantes, fonctions ou classes ont le même nom, tout plantera, pour la même raison qu'il ne peut y avoir deux fichiers portant le même nom dans un même dossier.

Comme vous l'aurez donc peut-être compris, le principe est de créer des dossiers puis de mettre la constante, fonction ou classe à l'intérieur. Ainsi, celle-ci pourra s'appeler comme elle le souhaitera, car elle sera dans un dossier à part, donc ne créera pas de conflit avec les autres qui sont à la racine. La seule différence majeure est qu'en PHP on n'appelle pas ça un **dossier**, mais un **namespace** (espace de noms). Comme pour les dossiers, vous pouvez en créer autant que vous voulez, il n'y a aucune limite. Vous pouvez placer à l'intérieur de chacun d'entre eux autant d'éléments que vous le souhaitez, à condition qu'il n'y ait pas deux constantes, fonctions ou classes du même nom dans le même namespace, cela va de soi.

Tout mon site a été réalisé sous une version antérieure à PHP 5.3. Je dois tout réécrire pour spécifier dans quel namespace mes éléments se trouvent ?

Bien sûr que non, encore heureux ! Actuellement, si vous n'avez rien changé, tous vos éléments sont dans l'espace de nom **global**. C'est un peu le dossier racine si vous voulez. Il est donc possible de réaliser tout son site sans spécifier de namespace.

L'utilisation des namespaces se fait donc pour éviter les conflits. Si vous utilisez différentes bibliothèques, il peut être intéressant de les encapsuler dans des namespaces afin que celles-ci n'entrent pas en conflit. Pour une petite application faite maison, cela n'est pas forcément très utile, mais à partir du moment où vous utilisez des constantes, fonctions ou classes d'un autre script, n'hésitez pas à les isoler. 

Déclarer un namespace

La déclaration d'un namespace se fait généralement dans un fichier à part. Il est déconseillé d'en déclarer plusieurs à l'intérieur d'un seul et même fichier, mais c'est possible.

Déclaration d'un namespace

Pour déclarer un namespace, il faut utiliser la commande **namespace** suivi du nom du namespace à créer. Rien ne doit être placé avant, que ce soit du HTML ou un simple retour à la ligne. Aucune instruction ne doit d'ailleurs être écrite avant la déclaration du namespace (sauf l'instruction **declare**). Bref, voici un exemple tout bête :

Code : PHP

```php
<?php
    namespace MonNamespace; // Déclaration du namespace.

    // Toutes les constantes, fonctions et classes déclarées ici
feront partie du namespace MonNamespace.
?>
```

Afin d'être sûr qu'on a créé un nouveau namespace, on va créer une fonction ayant le même nom qu'une autre fournie par PHP, puis exécuter le code.

Code : PHP

```php
<?php
    namespace MonNamespace;

    function strlen()
    {
        echo 'Hello world !';
    }
?>
```

Quand vous ouvrez cette page, aucune erreur n'est lancée ! La fonction `strlen()` a été créée **dans** notre nouveau namespace. Vous pouvez même l'appeler de cette façon :

Code : PHP

```php
<?php
    namespace MonNamespace;

    function strlen()
    {
        echo 'Hello world !';
    }
```

```
    strlen();
?>
```

Et vous verrez qu'à l'écran s'affichera *« Hello world ! »*.

Euh attends… Qu'est-ce qui s'est passé là ?

Ce qui s'est passé est très simple. Nous avons créé un nouveau namespace, et tout ce qui suit cette déclaration fait partie de ce namespace, c'est donc pour ça que ce n'est pas la fonction `strlen()` du namespace global qui a été appelée.

Et si je veux appeler une fonction du namespace global, je fais comment ?

Il vous suffit tout simplement de faire précéder le nom de la fonction d'un backslash (\), comme ceci :

Code : PHP

```
<?php
    namespace MonNamespace;

    function strlen()
    {
        echo 'Hello world !';
    }

    echo \strlen('Hello world !');
?>
```

Ce qui affiche bien « 13 ».

Si la constante ou fonction que vous avez appelée dans votre namespace n'existe pas, alors PHP ira chercher dans le namespace global. Si elle n'existe toujours pas dans le namespace global, alors une erreur sera levée. En fait, ce n'est pas toujours le cas, mais j'en reparlerai en temps voulu. Pour les classes, PHP lèvera directement une erreur si il ne la trouve pas dans le namespace actuel, il ne cherchera pas dans le namespace global !

Si vous incluez ce fichier (que l'on nomme par exemple **MonNamespace.ns.php**) dans un autre (par exemple **index.php**), alors le code contenu dans **index.php** fera partie du namespace global, et non de **MonNamespace** !

Déclaration de plusieurs namespaces dans un même fichier

C'est déconseillé, mais possible. Vous pouvez déclarer plusieurs namespaces à la suite, comme ceci :

Code : PHP

```
<?php
    namespace A;

    function quelNamespace()
    {
        echo 'A';
    }

    quelNamespace(); // Affiche « A ».

    namespace B;

    function quelNamespace()
```

```
        {
            echo 'B';
        }

        quelNamespace(); // Affiche « B ».
    ?>
```

Si vous décidez de déclarer plusieurs namespaces au sein d'un même fichier, il est préférable d'utiliser une autre méthode. En effet, mieux vaut utiliser les accolades :

Code : PHP

```
<?php
    namespace A
    {
        function quelNamespace()
        {
            echo 'A';
        }

        quelNamespace();
    }

    namespace B
    {
        function quelNamespace()
        {
            echo 'B';
        }

        quelNamespace();
    }
?>
```

Le résultat observé est le même. Sachez aussi qu'aucun code ne peut être écrit en dehors de ces accolades.

Et si je veux écrire du code dans le namespace global, je fais comment ?

Il vous suffit simplement d'ouvrir une nouvelle paire d'accolades, précédée du simple mot **namespace**, comme ceci :

Code : PHP

```
<?php
    namespace A
    {
        function quelNamespace()
        {
            echo 'A';
        }

        quelNamespace();
    }

    namespace B
    {
        function quelNamespace()
        {
            echo 'B';
        }

        quelNamespace();
```

```
    }

    namespace // Le code contenu dans ce namespace fera partie du
namespace global.
    {
        echo strlen('Hello world !');
    }
?>
```

Et à l'écran s'affichera *« AB13 »*. Tous les namespaces ont donc bien été exécutés, et le dernier fait bien partie du namespace global puisque `strlen()` a été correctement appelé.

Tu nous as dit que si une constante, fonction ou classe n'existait pas dans le namespace actuel, alors PHP irait la chercher dans le namespace global. Ta conclusion est donc un peu trop hâtive, non ?

Exact. Puisqu'il vous faut encore plus de preuves et que vous ne me croyez pas, je vais vous présenter une constante magique.

__NAMESPACE__

Cette constante magique est une chaîne de caractères qui contient le nom du namespace dans lequel on se trouve actuellement. Si on est dans l'espace de nom global, alors cette constante est vide.

Code : PHP

```
<?php
    namespace A
    {
        echo __NAMESPACE__;
    }

    namespace B
    {
     echo __NAMESPACE__;
    }

    namespace
    {
        echo __NAMESPACE__;
    }
?>
```

Et à l'écran s'affiche bien « *AB* ». Le dernier namespace est donc bien le namespace global. Convaincu ?

Pour vérifier mes dires, vous auriez aussi pu écrire une fonction `strlen()` dans le namespace global. Vous auriez ainsi observé une erreur indiquant que la fonction a déjà été déclarée. C'était plus simple, mais je voulais vous présenter la constante magique **__NAMESPACE__**.

Déclaration de sous-namespaces

Il est dommage de se limiter à la création de simple namespaces, au même titre que de simples dossiers à la racine. Il est donc possible, fort heureusement, de créer des **sous-namespaces** (vous pouvez en créer une infinité).

Pour créer un namespace à l'intérieur d'un autre, on écrira l'arborescence du namespace. Par arborescence j'entends tous les namespaces parents. Si je veux créer un namespace **B** dans **A**, alors **A** sera le namespace parent de **B**, et il va falloir le spécifier lors de la création de **B**.

Lorsque vous séparez deux namespaces, il faut les séparer par un backslash (le revoici, celui-là), comme les dossiers sous

Windows. Ainsi, si on veut créer **B** dans **A**, on fera comme ça :

Code : PHP

```php
<?php
    namespace A\B;

    echo __NAMESPACE__;
?>
```

Et à l'écran s'affiche... « *A\B* » ! En effet, quand on spécifie un namespace, on part toujours du premier namespace, puis on liste tous les sous-namespaces. Ainsi, le namespace **B** à proprement parler n'existe pas.

Notez que le namespace **A** est automatiquement créé si on décide de créer le namespace **A\B**. Le namespace **A** est donc vierge et on peut le remplir par la suite :

Code : PHP

```php
<?php
    namespace A\B
    {
        echo __NAMESPACE__;
    }

    namespace A
    {
        echo __NAMESPACE__;
    }
?>
```

À l'écran s'affichera bien « *A\BA* ».

Accéder à un élément d'un namespace différent

Créer des namespaces c'est bien, mais pouvoir appeler des constantes, fonctions ou classes d'un namespace différent, c'est encore mieux. Pour cela, il suffit de spécifier le namespace auquel appartient l'élément juste avant de l'appeler. Comme pour les fichiers, il y a deux façons de procéder.

- De façon **relative** : l'appel de l'élément dépendra du namespace dans lequel vous êtes. Celui-ci est **relatif** au namespace courant.
- De façon **absolue** : l'appel de l'élément se fait en partant du namespace global puis en listant tous les sous-namespaces jusqu'au namespace souhaité.

Prenons un exemple tout simple :

Code : PHP

```php
<?php
    namespace A\B
    {
        function getNamespace()
        {
            echo __NAMESPACE__;
        }
    }

    namespace A
    {
        B\getNamespace(); // Appel de façon relative.
        \A\B\getNamespace(); // Appel de façon absolue.
    }
```

```
?>
```

Vous revoyez encore ce backslash. Vous n'avez pas fini de le voir, dans le monde des namespaces sous PHP, c'est lui qui est utilisé pour séparer deux éléments.

Un peu de vocabulaire

Quand vous appelez une constante, fonction ou classe, alors le nom de l'élément que vous appelez est **qualifié** ou **non qualifié**.

- On dit que le nom de l'élément est qualifié si le namespace dans lequel il se trouve est spécifié. Exemple : `B\fonction()` et `\A\B\fonction()` sont tous les deux des éléments qualifiés.
- On dit que le nom de l'élément est non qualifié si le namespace dans lequel il se trouve n'est pas spécifié. Exemple : `fonction()` est un élément non qualifié.

Ceci m'oblige à revenir sur quelque chose que je vous ai dit plus tôt. Je vous avais dit que si la constante ou fonction appelée n'existait pas dans le namespace actuel, alors PHP irait chercher dans le namespace global. En fait, c'est le cas uniquement si le nom de l'élément est non qualifié. S'il est qualifié, alors PHP lèvera une erreur directement, sans aller chercher dans le namespace global. Pour rappel, si vous vous servez d'une classe dans un namespace dans lequel elle n'est pas déclarée, une erreur sera levée, que l'élément soit qualifié ou non.

Donc si je veux appeler un élément du namespace courant, celui-ci sera forcément un élément non qualifié, à moins que je l'appelle de façon absolue, c'est ça ?

Pas vraiment. Il existe un moyen d'appeler un élément du namespace courant de façon relative tout en étant qualifié. Pour cela, il y a le mot-clé **namespace**. Vous le connaissez déjà, c'est lui qui permet de déclarer un nouveau namespace, mais je vais vous présenter ici une autre de ces fonctionnalités : pouvoir appeler un élément de l'espace courant de façon relative tout en étant qualifié. Il s'utilise simplement comme ceci :

Code : PHP

```
<?php
    namespace A;

    echo strlen('Hello world !'); // Appel relatif, élément non qualifié.
    echo namespace\strlen('Hello world !'); // Appel relatif, élément qualifié. Résultat : erreur fatale car fonction inexistante.
?>
```

Comme vous l'aurez peut-être deviné, **namespace** représente le namespace actuel. Pour ceux qui savent programmer orienté objet, le mot-clé **namespace** peut être comparé au mot-clé **self**. L'un représente le namespace, l'autre la classe.

Créer des alias

Il est possible d'obtenir une arborescence assez longue. Par exemple, imaginez que vous devez utiliser le namespace A\B\C\D\E\F. L'écrire à chaque fois pour appeler une constante, fonction ou classe peut être lourd. Pour cela, vous pouvez dire à PHP *« À chaque fois que j'écrirai nsF **(par exemple)**, tu feras comme si j'avais écrit A\B\C\D\E\F »*. On dit dans ce cas-là que **nsF** est un alias de **A\B\C\D\E\F**.

Pour dire une telle chose à PHP, on utilise la commande **use**, suivie du long namespace (dans notre cas, il s'agit de A\B\C\D\E\F). Ensuite, il faut dire que le nom de l'alias est **nsF**. On va donc placer à la suite de cette commande le mot-clé **as** suivi du nom de l'alias (donc **nsF** dans notre cas).

Code : PHP

```
<?php
    use A\B\C\D\E\F as nsF;
?>
```

Testons notre code. Dans un fichier **F.ns.php**, placez ceci :

Code : PHP - F.ns.php

```php
<?php
    namespace A\B\C\D\E\F;

    function getNamespace()
    {
        echo __NAMESPACE__;
    }
?>
```

Dans un autre fichier quelconque, essayez le code suivant :

Code : PHP

```php
<?php
    require 'F.ns.php';

    use A\B\C\D\E\F as nsF;

    nsF\getNamespace(); // Se transforme en
A\B\C\D\E\F\getNamespace().
?>
```

Et à l'écran s'affiche bien « *A\B\C\D\E\F* ». On a donc bien appelé la fonction getNamespace() du namespace **A\B\C\D\E\F**.

Vous pouvez créer plusieurs alias en les séparant par une virgule, comme ceci :

Code : PHP

```php
<?php
    use A\B\C\D\E\F as nsF, G\H\I\J\K\L as nsL;
?>
```

Il existe aussi une autre façon d'utiliser **use**. En effet, vous n'êtes pas obligé de spécifier le nom de l'alias avec **as**. Dans ce dernier cas, le nom de l'alias sera le nom du dernier namespace. Exemple :

Code : PHP

```php
<?php
    use A\B\C\D\E\F;
    // …revient au même que d'écrire :
    use A\B\C\D\E\F as F;
?>
```

Afin de vous convaincre, vous n'avez qu'à utiliser ce code :

Code : PHP

```php
<?php
```

```php
    require 'F.ns.php';

    use A\B\C\D\E\F;

    F\getNamespace(); // Se transforme en
A\B\C\D\E\F\getNamespace().
?>
```

Importation de classes

Nous allons voir une façon d'utiliser **use** qui permet d'importer des classes, et **uniquement** des classes (ça ne fonctionnera pas avec des constantes ou fonctions). Le but est d'importer la classe d'un certain namespace dans le namespace courant. Par exemple, dans le fichier F.ns.php, nous allons créer une classe toute bête :

Code : PHP - F.ns.php

```php
<?php
    namespace A\B\C\D\E\F;

    class MaClasse
    {
        public function hello()
        {
            echo 'Hello world !';
        }
    }
?>
```

Pour importer la classe, il suffit de taper **use espace\de\noms\MaClasse**. Exemple :

Code : PHP

```php
<?php
    require 'F.ns.php';

    use A\B\C\D\E\F\MaClasse;

    $a = new MaClasse; // Se transforme en $a = new
A\B\C\D\E\F\MaClasse.
    $a->hello();
?>
```

Tout fonctionne bien, à l'écran s'affichera bien « *Hello world !* ». Vous pouvez bien entendu utiliser le mot-clé **as** afin de modifier le nom de la classe :

Code : PHP

```php
<?php
    require 'F.ns.php';

    use A\B\C\D\E\F\MaClasse as Hello;

    $a = new Hello; // Se transforme en $a = new
A\B\C\D\E\F\MaClasse.
    $a->hello();
?>
```

Ce tutoriel s'arrête ici. J'espère qu'il aura été utile et agréable à lire.

Ce tutoriel a été corrigé par les zCorrecteurs.