Espacios de nombres en PHP (I). Introducción.

Todos hemos oído hablar alguna vez de los espacios de nombres, o namespaces (ns), en terminología anglosajona. Tal vez en otro contexto (cómo XML, por ejemplo). En PHP constituyen una valiosa herramienta de la que vamos a ocuparnos en estos artículos. En su definición más aceptada, son una forma de encapsular elementos (en el contexto de PHP, estamos hablando de constantes, funciones y clases).
Esto nos facilita crear código reutilizable, permitiendo a otros desarrolladores implementar con facilidad nuestras librerías, evitando colisiones de nombres que pudieran surgir con sus propios elementos, y viceversa. Nosotros podemos implementar librerías de terceros, con la misma prerrogativa.

QUÉ SON LOS ESPACIOS DE NOMBRES
Para aquellos de vosotros que no estéis familiarizados con esta técnica, debemos concebir los espacios de nombres con respecto a las constantes, clases y funciones como los directorios con respecto a los ficheros en un soporte de almacenamiento. No es posible tener dos ficheros con el mismo nombre en el mismo directorio, pero no hay ningún problema con tener dos ficheros con el mismo nombre en directorios diferentes. De esta forma, el sistema operativo siempre podrá almacenar estos ficheros sin problemas, y también podremos recuperarlos, precediendo el nombre de la ruta en la que se encuentran. Cómo ya sabes, esto funciona así sea cual sea tu sistema operativo.
Con los elementos de PHP ocurre lo mismo. No podemos tener en un script, por ejemplo, dos funciones con el mismo nombre. Sin embargo, si cada una de ellas está encapsulada en un espacio de nombres diferente, no surge ninguna colisión. A lo largo de estos artículos vamos a aprender a crear y emplear namespaces para organizr y encapsular nuestros elementos.
UN EJEMPLO SIMPLE
Vamos a empezar viendo un uso simple de los espacios de nombres, para comprender cómo operan. Tenemos un script muy sencillo, llamado s1.php, con el siguiente listado:

```
<?php
 namespace ns1;

 const CONSTANTE = 'Valor de constante en ns1'.'<br>';

 function funcionDePrueba(){
  echo "Esta en la función del script en ns1".'<br>';
 }

 class ClaseDePrueba{
  public static function metodoDeClase(){
   echo "Esta es una función de clase en ns1".'<br>';
  }
 }
?>
```

Como ves, tenemos una constante, una función y una clase. No es que sean gran cosa, ni hagan nada útil, pero, a los efectos didácticos, es todo lo que necesitamos.
Ahora observa la primera línea (resaltada en el listado):

```
namespace ns1;
```

Lo que hacemos es usar la instrucción namespace para definir un espacio de nombres al que hemos llamado ns1. Todos los elementos que hay en este script pertenecen ahora a este espacio de nombres (en seguida veremos como usarlo).

ATENCIÓN. La instrucción namespace, cuando se refiere a todos los elementos que aparecen definidos en el script, debe estar, indefectiblemente, al principio de dicho script. En esta serie de artículos veremos otros modos de uso pero, por ahora, grábate a fuego esta máxima como dogma de fe.

ATENCIÓN. El uso de define() para declarar constantes provoca problemas cuando se emplean namespaces. En su lugar, es mejor declararlas con la instrucción const, como vemos en el ejemplo.

Ahora vamos a ver otro script muy parecido, llamado s2.php:

```
<?php
 namespace ns2;

 const CONSTANTE = 'Valor de constante en ns2'.'<br>';

 function funcionDePrueba(){
  echo "Esta en la función del script en ns2".'<br>';
 }

 class ClaseDePrueba{
  public static function metodoDeClase(){
   echo "Esta es una función de clase en ns2".'<br>';
  }
 }
?>
```

Como ves, este tiene tres elementos, cuyos nombres coinciden con los que hay en s1.php, aunque el valor de la constante, y lo que podemos obtener con la función y la clase es diferente en ambos. También cambia el nombre del namespace que, en este es, como ves, ns2. Bajo este espacio de nombres se encapsulan los elementos de este script. Me imagino que ya ves por donde sopla el viento.
Tenemos un script principal al que hemos llamado index.php, con el siguiente listado:

```
<?php
 include 's1.php';
 include 's2.php';

 const CONSTANTE = 'Valor de constante en script principal'.'<br>';

 function funcionDePrueba(){
  echo "Esta en la función del script en script principal".'<br>';
 }

 class ClaseDePrueba{
  public static function metodoDeClase(){
   echo "Esta es una función de clase en script principal".'<br>';
  }
 }
```

```
 echo CONSTANTE;
 funcionDePrueba();
 ClaseDePrueba::metodoDeClase();

 echo ns1CONSTANTE;
 ns1funcionDePrueba();
 ns1ClaseDePrueba::metodoDeClase();

 echo ns2CONSTANTE;
 ns2funcionDePrueba();
 ns2ClaseDePrueba::metodoDeClase();
?>
```

Observa que lo primero que hacemos es incluir los otros dos scripts, además de declara, a continuación, una constante, una función y una clase con los mismos nombres que ya tenemos. Si no estuviéramos usando espacios de nombres, PHP entendería que estamos tratando de redeclarar estos elementos (lo que se llama una colisión de nombres) y abortaría la ejecución con un mensaje de error:
Fatal error: Cannot redeclare funcionDePrueba() (previously declared in C:xampphtdocsnsindex.php:8) in **C:xampphtdocsnss1.php** on line **8**
Sin embargo, al usar espacios de nombres, los elementos de cada script se hallan perfectamente encapsulados bajo sus respectivos namespaces. El resultado de ejecutar index.php bajo estas circunstancias es el siguiente:

```
Valor de constante en script principal
 Esta en la función del script en script principal
 Esta es una función de clase en script principal
 Valor de constante en ns1
 Esta en la función del script en ns1
 Esta es una función de clase en ns1
 Valor de constante en ns2
 Esta en la función del script en ns2
 Esta es una función de clase en ns2
```

Observa, en las líneas resaltadas, como referenciamos los distintos elementos. Los del propio script index.php, que no se hallan, en este ejemplo, bajo ningún espacio de nombres, los referenciamos directamente, cómo hacemos de la forma tradicional. Los elementos que pertenecen a los otros scripts incluidos los referenciamos precediéndolos del correspondiente espacio de nombres y la barra invertida. Sólo con esto ya hemos eliminado cualquier ambigüedad y, por ende, cualquier colisión.

ATENCIÓN. El separar el espacio de nombres del nombre del elemento mediante una barra invertida () es la razón por la que no podemos emplear esta técnica con variables. Sabemos que las variables en PHP se preceden con el signo $. Si empleásemos un espacio de nombres, la barra invertida estaría escapando el signo $, con lo que no podríamos referenciar el nombre de la variable. Sin embargo, encapsular variables en PHP no es ningún problema ya que siempre podemos declararlas dentro de una clase (como establece el paradigma POO) y hacerlas privadas. Y la clase sí podemos encapsularla, como has visto, con espacios de nombres.

Y, por ahora, eso es todo. En el próximo artículo seguiremos aprendiendo sobre espacios de nombres en PHP. Los scripts de este artículo te los dejo en este enlace.