PHP - Namespaces

We often organize the files in different folders. Usually a folder contains files related to a certain objective, or application or category. A folder can't contain two files with the same name, though different folders may have a file of the same name so that the path of each file is different.

The idea of namespaces in PHP is somewhat similar. In PHP, namespaces allow classes or functions or constants of same name be used in different contexts without any conflict, thereby encapsulating these items.

A PHP namespace is logical grouping of classes/functions etc., depending on their relevance. Just as a file with same name can exist in two different folders, a class of a certain name can be defined in two namespaces. Further, as we specify the complete path of a file to gain access, we need to specify full name of class along with namespace.

As your application size becomes bigger, involving many class and function definitions, giving give a unique name to each class/function may become tedious and not exactly elegant. Using namespaces lets you organize such code blocks in a neat manner. For example, if we need to declare a calculate() function to calculate area as well as tax, instead of defining them as something like calculate_area() and calculate_tax(), we can create two namespaces area and tax and use calculate() inside them.

Advantages of Namespace

Here are some of the advantages of using namespaces in PHP –

- Namsepaces help in avoiding name collisions between classes/functions/constants defined by someone with third-party classes/functions/constants.
- Namespaces provide the ability to alias (or shorten) Extra_Long_Names, thereby improving the readability of source code.
- PHP Namespaces provide a way in which to group related classes, interfaces, functions and constants. Namespace names are case – insensitive.

Defining a Namespace

PHP's namespace keyword is used to define a new namespace.

```
namespace myspace;
```

A ".php" file containing a namespace must declare the namespace at the top of the file before any other (except the declare directive). Declaration of class, function and constants inside a namespace affects its access.

A PHP script may contain other code apart from the definition of a namespace. To load the namespace defined in the same code, PHP has the "use" keyword.

```
use myspace;
```

Example

In the following "hello.php" script, we define a hello() function inside myspace namespace, and call it after loading the namespace in the current script.

</> Open Compiler

```
<?php
   namespace myspace;
   function hello() {
      echo "Hello World";
   }
   use myspace;
   myspace\hello();
?>
```

It will produce the following **output** –

```
Hello World
```

Note that you must qualify the hello() function with its full name that includes the namespace - myspace\hello().

Include Namespace

You may have one script consisting of a declaration of a namespace, and the other script in which the namespace is loaded with include statement.

a.php

```php
<?php
   namespace myspace {
      function hello() {
         echo "Hello World in myspace";
      }
   }
?>
```

b.php

```php
<?php
   include 'a.php';
   myspace\hello();
?>
```

It will produce the following **output** –

```
Hello World in myspace
```

There may be a case where the current script ("b.php" as above) also has a function of the same name as in the included file. The fully qualified function that prepends the namespace, helps the parser to resolve the name conflict.

Example

Take a look at the following example –

```php
<?php
   include 'a.php';
   function hello() {
      echo "Hello World from current namespace";
   }
   hello();
   myspace\hello();
?>
```

It will produce the following **output** –

```
Hello World from current namespace
Hello World in myspace
```

Example

As mentioned above, the namespace declaration must be at the top, immediately after the opening <?php tag. Otherwise the parser throws a fatal error.

</> Open Compiler

```
<?php
   echo "hello"
   namespace myspace;
   function hello() {
      echo "Hello World";
   }
   use myspace;
   myspace\hello();
?>
```

It will produce the following **output** –

```
PHP Parse error:  syntax error, unexpected token "namespace",
expecting "," or ";" in /home/cg/root/67771/main.php on line 4
```

The above error message makes it clear that only the "declare statement" is allowed to appear before the namespace declaration.

</> Open Compiler

```
<?php
   declare (strict_types=1);
   namespace myspace;
   function hello() {
      echo "Hello World";
   }
   use myspace;
   myspace\hello();
?>
```

Relative Namespace

The objects such as functions, classes and constants may be accessed in the current namespace by referring the with relative namespace paths.

In the following example, "b.php" contains a namespace space1\myspace with a hello() function and a TEMP constant. The same objects are also defined in namespace space1, present in "a.php".

Obviously, when "b.php" is included in "a.php", "myspace" is a subspace of "space1". Hence, hello() from "myspace" is called by prefixing its relative namespace (also the TEMP constant)

b.php

```
<?php
   namespace space1\myspace;
   const TEMP = 10;
   function hello() {
      echo "Hello from current namespace:" . __NAMESPACE__ . ;
   }
?>
```

a.php

```
<?php
   namespace space1;
   include 'b.php';
   function hello() {
      echo "Hello from current namespace:" . __NAMESPACE__ . ;
   }
   const TEMP = 100;
   hello();            // current namespace
   myspace\hello();   // sub namespace

   echo "TEMP : " . TEMP . " in " . __NAMESPACE__ . ;
   echo "TEMP : " . myspace\TEMP  . " \\in space1\\myspace\n";
?>
```

It will produce the following **output** –

```
Hello from current namespace:space1
Hello from current namespace:space1\myspace
```

```
TEMP : 100 in space1
TEMP : 10 in space1\myspace
```

Absolute Namespace

You can also access the functions/constants from any namespace by prefixing the absolute namespace path. For example, hello() in "b.php" is "\space\myspace\hello()".

a.php

```php
<?php
   namespace space1;
   include 'b.php';
   function hello() {
      echo "Hello from current namespace:" . __NAMESPACE__ . ;
   }
   const TEMP = 100;
   \space1\hello();              //current namespace
   \space1\myspace\hello();     //sub namespace

   echo "TEMP: " . \space1\TEMP . " in " . __NAMESPACE__ . ;
   echo "TEMP: " . \space1\myspace\TEMP  . " in space1\\myspace\n";
?>
```

The __NAMESPACE__ is a predefined constant in PHP that returns the name of current namespace.

Namespace Rules

Any conflict in the names of function/classes/constants appearing between different namespaces is resolved by following these rules –

- A namespace identifier without namespace separator symbol (/) means it is referring to current namespace. This is an unqualified name.
- If it contains separator symbol as in myspace\space1, it resolves to a subnamespace space1 under myspace. Such type of naming is relative namespace.
- Name of fully qualified namespace starts with the "\" character. For example, "\myspace" or "\myspace\space1".
- Fully qualified names resolve to absolute namespace. For example \myspace\space1 resolves to myspace\space1 namespace

- If the name occurs in the global namespace, the "namespace\" prefix is removed. For example, "namespace\space1" resolves to space1.
- However, if it occurs inside another namespace, it is treated differently. For example, if namespace\space1 is inside myspace, it is equivalent to "myspace\space1".
- First segment of the name in qualified name is translated according to the current class/namespace import table.
- If no import rule applies, the current namespace is prepended to the name.
- class-like names are translated according to the class/namespace import table, function names according to the function import table and constants according to the constant import table.
- For unqualified names, if no import rule applies and the name refers to a function or constant and the code is outside the global namespace, the name is resolved at runtime. First it looks for a function from the current namespace, then it tries to find and call the global function.