JUNTA DE EXTREMADURA

Consejería de Hacienda y Administración Pública

Dirección General de Tecnologías de la Información y Comunicación

MADES

Convenio de codificación en PHP

06/03/2018

HOJA DE CONTROL

Proyecto	MADES (Marco para el Desarrollo y Mantenimiento de los S.I.)		
Documento	Convenio de codificación en PHP		
Nombre del Fichero	MADES - Convenio de codificación en PHP.odt		
Autor			
Versión/Edición	1.0	**Fecha Versión**	06/03/2018
Aprobado por		**Fecha Aprobación**	

REGISTRO DE CAMBIOS

Versión	Causa del Cambio	Responsable del Cambio	Área	Fecha del Cambio

CONTROL DE DISTRIBUCIÓN

Nombre y Apellidos	Cargo	Área	Nº Copias

ÍNDICE

INTRODUCCIÓN

Objetivo

El objetivo del documento es definir las reglas de formato para los archivos de código fuente en PHP. Su objetivo es el de mejorar la colaboración, la calidad del código y habilitar la infraestructura de apoyo.

Alcance

Todos los proyectos de desarrollo y mantenimiento de sistemas de información en lenguaje PHP de la DGTIC.

Ámbito de Responsabilidades

De aplicación a los proveedores y responsables de proyectos de sistemas de información de la DGTIC, así como por la unidad responsable de calidad que velará por su cumplimiento en cada sistema o aplicación.

Revisión de la Norma

La norma que describe el Marco de Desarrollo de Sistemas de Información de la DGTIC, desarrollada en este documento será revisada de forma anual, para adaptar, corregir y ampliar en su caso tanto su alcance como su contenido. La unidad competente de la DGTIC será quien determine los responsables de la revisión, adaptación y modificación de la norma.

Publicación de la Norma

El presente documento, así como las revisiones, serán de dominio público y se pondrá a disposición de cualquier ciudadano interesado utilizando los medios que la DGTIC estime oportunos, preferentemente a través de Internet, desde los portales de información que la Junta de Extremadura tiene de acceso público.

PSR-1: ESTÁNDAR DE CÓDIGO BÁSICO

Esta sección del estándar comprende los que deberían ser considerados como los elementos de código estándar que son necesarios para asegurar un alto nivel de interoperabilidad técnica entre el código PHP compartido.

1.1 General

- Los ficheros DEBEN usar sólo las etiquetas <?php y <?=.
- Los ficheros DEBEN usar sólo codificación UTF-8 sin BOM (byte order mark) en el código PHP.
- Los ficheros DEBERÍAN declarar símbolos (clases, funciones, constantes, etc.) o causar efectos secundarios (p.ej: generar una salida, cambiar configuraciones .ini, etc.) pero NO DEBERÍAN hacer ambas cosas.
- Los espacios de nombre y las clases DEBEN cumplir el PSR de "autocarga" (PSR-4).
- Los nombres de clase DEBEN declararse en StudlyCaps, normalmente en CamelCase, comenzando en mayúscula.
- Las constantes de clase DEBEN declararse completa en mayúsculas separando las palabras con subrayado o guión bajo.
- Los nombres de métodos se DEBEN declarar en camelCase, comenzando en minúscula.

1.2 Ficheros

Etiquetas PHP

El código en PHP DEBE usar las etiquetas largas <?php ?> o las etiquetas de echo corto <?= ?>; NO DEBE usar otras variaciones de etiquetas.

Codificación de Caracteres

El código en PHP DEBE usar únicamente UTF-8 sin BOM.

Efectos Secundarios

Un fichero DEBERÍA declarar nuevos símbolos (clases, funciones, constantes, etc.) y no causar otros efectos secundarios, o DEBERÍA ejecutar lógica con efectos secundarios, pero NO DEBERÍA hacer ambas cosas.

La frase "efectos secundarios" hace referencia a la ejecución de lógica no relacionada directamente con la declaración de clases, constantes, funciones, etc. sino meramente con la inclusión del fichero.

Los "efectos secundarios" incluyen pero no están limitados a: generación de salida, uso explícito de require o include, conexion con servicios externos, modificación de configuraciones ini, emisión de errores o excepciones, modificación de variables globales o estáticas, leer o escribir de un fichero y similares.

Lo siguiente es un ejemplo de un fichero tanto con declaraciones como efectos secundarios; es decir, un ejemplo de qué evitar:

```
<?php
// efecto secundario: cambio de configuración ini
ini_set('error_reporting', E_ALL);

// efecto secundario: cargar un fichero
include "file.php";

// efecto secundario: generar salida
echo "<html>\n";

// declaración
function foo()
{
    // cuerpo de la función
}
```

El siguiente ejemplo es de un fichero que contiene declaraciones sin efectos secundarios; es decir, un ejemplo a emular:

```php
<?php
// declaración
function foo()
{
    // cuerpo de la función
}

// una declaración condicional *no* es un efecto secundario
if (! function_exists('bar')) {
    function bar()
    {
        // cuerpo de la función
    }
}
```

1.3 Espacios de nombre y Nombres de Clase

Los espacios de nombre y las clases DEBEN seguir la recomendación PSR de "autocarga" [PSR-4].

Esto significa que cada clase se encuentra contenida en un fichero y en un espacio de nombre de al menos un nivel: un nombre de proveedor de nivel superior.

Los nombres de clase DEBEN declararse en formato *StudlyCaps*, normalmente en CamelCase comenzando con mayúsculas.

El código escrito para PHP 5.3 y versiones posteriores DEBE utilizar espacios de nombre formales. Por ejemplo:

```php
<?php
// PHP 5.3 y siguientes:
namespace Vendor\Model;

class Foo
{
}
```

El código escrito para la versión 5.2.x y anteriores DEBERÍAN utilizar convención de pseudo-espaciado de nombres de prefijos de proveedor en los nombres de clase.

```php
<?php
// PHP 5.2.x y anteriores:
class Vendor_Model_Foo
{
}
```

1.4 Constantes de Clase, Propiedades y Métodos

El término "clase" hace referencia al conjunto de clases, interfaces y rasgos (traits).

Constantes

Las constantes de clase DEBEN declararse en mayúsculas separando las palabras con guión bajo. Por ejemplo:

```php
<?php
namespace Vendor\Model;

class Foo
{
    const VERSION = '1.0';
    const DATE_APPROVED = '2012-06-01';
}
```

Propiedades y Métodos

Tanto los nombres de propiedades como los de métodos DEBEN tener el formato camelCase, es decir, palabras capitalizadas, sin espacios y con la primera letra en minúscula.

Aunque en el estándar original PSR 1, se evita cualquier recomendación en el nombre de las propiedades, en el definido en este documento se prefiere optar por el definido en el apartado anterior.

PSR-2: GUÍA DE ESTILO DE CODIFICACIÓN

Esta guía extiende la recomendación PSR-1, el estándar de código básico.

La intención de esta guía es reducir la fricción cognitiva cuando se lee código de diferentes autores. Lo hace enumerando un conjunto compartido de reglas y expectativas sobre cómo dar formato al código en PHP.

Las reglas de estilo de este documento se derivan de las similitudes entre los diversos proyectos de los miembros del grupo de frameworks. Cuando varios autores colaboran a través de múltiples proyectos, esta guía ayuda a tener un conjunto de directrices para ser utilizado entre todos esos proyectos. Por lo tanto, el beneficio de esta guía no está en las reglas mismas, sino en compartir esas reglas.

2.1 Resumen

- El código DEBE seguir una "guía de estilo de codificación" PSR [PSR-1].
- El código DEBE utilizar 4 espacios para indentar, no tabulaciones.
- NO DEBE haber un límite fijo en la longitud de línea; el límite aproximado DEBE ser de 120 caracteres; la líneas DEBERÍAN ser de 80 caracteres o menos.
- DEBE haber una línea en blanco después de la declaración de espacio de nombre, y DEBE haber una línea en blanco después del bloque de declaración use.
- Las llaves de apertura para las clases DEBEN ir en la línea siguiente, y las llaves de cierre DEBEN ir en la siguiente línea después del cuerpo de la clase.
- Las llaves de apertura de los métodos DEBEN ir en la siguiente línea, y las llaves de cierre DEBEN ir en la siguiente línea después del cuerpo del método.
- La visibilidad DEBE ser declarada en todas las propiedades o métodos; abstract y final DEBEN ser declaradas antes de la visibilidad; static DEBE ser declarada después de la visibilidad.
- Las palabras clave de las estructuras de control DEBEN tener un espacio después de ellas; las llamadas a los métodos y las funciones NO DEBEN tenerlo.

- Las llaves de apertura de las estructuras de control DEBEN ir en la misma línea, y las de cierre DEBEN ir en la línea siguiente al cuerpo de la estructura.

- Los paréntesis de apertura para las estructuras de control NO DEBEN tener un espacio después de ellos, y los de cierre NO DEBEN tener un espacio antes de de ellos.

Ejemplo

Este ejemplo resume algunas de las reglas citadas:

```php
<?php
namespace Vendor\Package;

use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class Foo extends Bar implements FooInterface
{
    public function sampleMethod($a, $b = null)
    {
        if ($a === $b) {
            bar();
        } elseif ($a > $b) {
            $foo->bar($arg1);
        } else {
            BazClass::bar($arg2, $arg3);
        }
    }

    final public static function bar()
    {
        // cuerpo del método
    }
}
```

2.2 General

Estándar de Codificación Básico

El código DEBE seguir todas las reglas expuestas en PSR-1.

Ficheros

- Todos los ficheros PHP DEBEN usar el final de fichero de UNIX LF (*linefeed*).

- Todos los ficheros PHP DEBEN terminar con una única línea en blanco.
- La etiqueta de cierre ?> se DEBE omitir de aquellos ficheros que solo contienen código en PHP.

Líneas

- No DEBE haber un límite fijo en la longitud de la línea de código.
- El límite aproximado DEBE ser de 120 caracteres; los correctores de estilo automáticos DEBEN avisar pero NO DEBEN provocar un error en el límite aproximado.
- Las líneas NO DEBERÍAN ser superiores a los 80 caracteres; las líneas más largas DEBERÍAN ser divididas en múltiples líneas a continuación de no más de 80 caracteres cada una.
- NO DEBE haber espacios en blanco al final de las líneas no vacías.
- Se PUEDEN añadir líneas en blanco para mejorar la legibilidad e indicar bloques relacionados de código.
- NO DEBE haber más de una instrucción por línea.

Sangría

- El código DEBE usar una sangría de 4 espacios, y NO DEBE utilizar tabulaciones.

Nota: Utilizar únicamente espacios, y no mezclar espacios con tabulaciones, ayuda a evitar los problemas que aparecen al comparar ficheros (diff), parches, históricos y anotaciones. El uso de espacios también hace que sea fácil insertar sangrías de más bajo nivel con más detalle para la alineación entre líneas.

Palabras clave y True/False/Null

- Las palabras clave de PHP DEBEN escribirse en minúsculas.
- Las constantes de PHP true, false y null DEBEN escribirse en minúsculas.

2.3 Espacio de Nombre y Declaraciones use

- Cuando esté presente, DEBE haber una línea en blanco después de la declaración del espacio de nombre.
- Cuando esté presente, todas las declaraciones use DEBEN ir después de la declaración del espacio de nombre.

- DEBE haber una palabra clave use por declaración.
- DEBE haber una línea en blanco después del bloque de declaraciones *use*.

Por ejemplo:

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

// ... código PHP adicional ...
```

2.4 Clases, Propiedades y Métodos

El término "clase" hace referencia al conjunto de clases, interfaces y rasgos (traits).

Extends e Implements

Las palabras clave extends e implements DEBEN declararse en la misma línea que el nombre de la clase.

La llave de apertura de la clase DEBE ir en la siguiente línea; la llave de cierre de la clase DEBE ir en la línea siguiente después del cuerpo de la clase.

```php
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // constantes, propiedades, métodos
}
```

La lista de implements PUEDE ser dividida en múltiples líneas, de forma que cada línea subsecuente esté sangrada una vez. Cuando se haga así, el primer elemento de la lista DEBE ir en la siguiente línea y DEBE haber solo un *interface* por línea.

```php
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // constantes, propiedades, métodos
}
```

Propiedades

- Se DEBE declarar la visibilidad de todas las propiedades de la clase.
- La palabra clave var NO DEBE usarse para declarar una propiedad.
- NO DEBE haber más de una propiedad por declaración.
- Los nombres de propiedades NO DEBERÍAN tener un prefijo con un carácter de subrayado para indicar la visibilidad protected o private.

Una declaración de propiedad se parece a lo siguiente:

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public $foo = null;
}
```

Métodos

- Se DEBE declarar la visibilidad de todos los métodos.
- Los nombres de métodos NO DEBERÍAN tener un prefijo con un carácter de subrayado para indicar la visibilidad protected o private.
- Los métodos NO DEBEN declararse con un espacio después del nombre del método.
- La llave de apertura DEBE ir en su propia línea y la llave de cierre DEBE ir en la línea siguiente después del cuerpo del método.
- NO DEBE haber un espacio después del paréntesis de apertura y NO DEBE haber un espacio antes del paréntesis de cierre.

Una declaración de método se parece a la siguiente. Hay que fijarse en la colocación de los paréntesis, comas, espacios y llaves:

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function fooBarBaz($arg1, &$arg2, $arg3 = [])
    {
        // cuerpo del método
    }
}
```

Argumentos de Método

- En la lista de argumentos, NO DEBE haber un espacio antes de cada coma y DEBE haber un espacio después de cada coma.
- Los argumentos de los métodos con valores por defecto DEBEN ir al final de la lista de argumentos.

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function foo($arg1, &$arg2, $arg3 = [])
    {
        // cuerpo del método
    }
}
```

La lista de argumentos PUEDE ser dividida en múltiples líneas, donde cada línea subsecuente se sangra una vez. Cuando se haga así, el primer elemento de la lista DEBE estar en la siguiente línea y DEBE haber solo un argumento por línea.

Cuando la lista de argumentos se divida en múltiples líneas, el paréntesis de cierre y la llave de apertura DEBEN estar colocados juntos, en su propia línea y con un espacio entre ellos.

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    ) {
        // cuerpo del método
    }
}
```

abstract, final y static

- Cuando estén presentes, las declaraciones abstract y final DEBEN preceder la declaración de visibilidad.

- Cuando esté presente, la declaración static DEBE ir después de la declaración de visibilidad.

```
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // cuerpo del método
    }
}
```

Llamadas a Métodos y Funciones

Cuando se haga una llamada a un método o una función, NO DEBE haber un espacio entre el nombre del método o función y el paréntesis de apertura; NO DEBE haber un espacio después del paréntesis de apertura y NO DEBE haber un espacio antes del paréntesis de cierre. En la lista de argumentos, NO DEBE haber un espacio antes de cada coma y DEBE existir un espacio después de cada coma.

```
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```

Las listas de argumentos PUEDEN ser divididas en múltiples líneas, donde cada línea subsecuente se sangra una vez. Cuando se haga así, el primer elemento de la lista DEBE estar en la siguiente línea y DEBE haber un único argumento por línea.

```
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```

2.5 Estructuras de Control

Las reglas de estilo general para las estructuras de control son las siguientes:

- DEBE haber un espacio después de la palabra clave de la estructura de control
- NO DEBE haber un espacio después del paréntesis de apertura
- NO DEBE existir un espacio antes del paréntesis de cierre
- DEBE haber un espacio entre el paréntesis de cierre y la llave de apertura
- El cuerpo de la estructura se DEBE sangrar una vez
- La llave de cierre DEBE estar en la línea siguiente después del cuerpo de la estructura
- El cuerpo de cada estructura DEBE estar encerrado entre llaves. Esto estandariza como aparecen las estructuras y reduce la probabilidad de introducir errores a medida que se añaden nuevas líneas al cuerpo.

if, elseif, else

Una estructura if se muestra como la siguiente. Fíjese en la localización de los paréntesis, los espacios y las llaves; y que las sentencias else y elseif están en la misma línea que la llave de cierre del cuerpo precedente.

```php
<?php
if ($expr1) {
    // cuerpo if
} elseif ($expr2) {
    // cuerpo elseif
} else {
    // cuerpo else;
}
```

Se DEBERÍA usar la palabra clave elseif en lugar de else if para que todas las palabras clave de control aparezcan como una única palabra.

switch, case

Una estructura switch es como la siguiente. Hay que fijarse en la localización de los paréntesis, los espacios y las llaves. Las sentencias case DEBEN estar sangradas una vez desde el switch; la palabra clave break (o cualquier sentencia final del case) DEBE estar sangrada al mismo nivel que el cuerpo de la palabra case. DEBE haber un comentario como "//no break" cuando la ejecución hacia abajo sea intencionada en un cuerpo case no vacío.

```php
<?php
switch ($expr) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
        return;
    default:
        echo 'Default case';
        break;
}
```

while, do while

Una sentencia while es como la siguiente. Hay que fijarse en la colocación de paréntesis, espacios y llaves.

```php
<?php
while ($expr) {
    // cuerpo de la estructura
}
```

De forma similar, una sentencia do while es como la siguiente.

```php
<?php
do {
    // cuerpo de la estructura
} while ($expr);
```

for

Una sentencia for es como la siguiente. Hay que fijarse en la colocación de paréntesis, espacios y llaves.

```php
<?php
for ($i = 0; $i < 10; $i++) {
    // cuerpo del for
}
```

foreach

Una sentencia foreach es como la siguiente. Hay que fijarse en la colocación de paréntesis, espacios y llaves.

```php
<?php
foreach ($iterable as $key => $value) {
    // cuerpo del foreach
}
```

try, catch

Un bloque try catch es como el siguiente. Hay que fijarse en la colocación de paréntesis, espacios y llaves.

```php
<?php
try {
    // cuerpo del try
} catch (FirstExceptionType $e) {
    // cuerpo del catch
} catch (OtherExceptionType $e) {
    // cuerpo del catch
}
```

2.6 Funciones Anónimas o Closures

- Las funciones anónimas DEBEN declararse con un espacio después de la palabra clave function y con un espacio antes y después de la palabra clave use.
- La llave de apertura DEBE ir en la misma línea y la de cierre DEBE ir en la línea siguiente después del cuerpo.
- NO DEBE haber un espacio después del paréntesis de apertura de la lista de argumentos o lista de variables; y NO DEBE haber un espacio antes del paréntesis de cierre de la lista de argumentos o lista variables.
- En la lista de argumentos y en la lista de variables, NO DEBE haber un espacio antes de cada coma y DEBE haber un espacio después de cada coma.
- Las funciones anónimas con valores por defecto DEBEN tener éstos colocados al final de la lista de argumentos.

Una declaración de función anónima es como la siguiente. Hay que fijarse en la colocación de paréntesis, comas, espacios y llaves:

```php
<?php
$closureWithArgs = function ($arg1, $arg2) {
    // cuerpo
};

$closureWithArgsAndVars = function ($arg1, $arg2) use ($var1, $var2) {
    // cuerpo
};
```

Las listas de argumentos y las listas de variables PUEDEN ser divididas en múltiples líneas, donde cada línea subsecuente se sangra una vez. Cuando se haga así, el primer elemento de la lista DEBE estar en la siguiente línea y DEBE haber solo un argumento o variable por línea.

Cuando la lista final (ya sean argumentos o variables) está dividida en múltiples líneas, el paréntesis de cierre y la llave de apertura DEBEN estar colocados juntos en su propia línea con un espacio entre ellos.

Los siguientes son ejemplos de funciones anónimas con y sin listas de argumentos y variables divididas en múltiples líneas.

```php
<?php
$longArgs_noVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) {
    // cuerpo
};

$noArgs_longVars = function () use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // cuerpo
};

$longArgs_longVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // cuerpo
};

$longArgs_shortVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use ($var1) {
    // cuerpo
};

$shortArgs_longVars = function ($arg) use (
```

```
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
    // cuerpo
};
```

Hay que fijarse que las reglas de formato también se aplican cuando la función anónima se utiliza directamente como un argumento en la llamada de una función o método.

```
<?php
$foo->bar(
    $arg1,
    function ($arg2) use ($var1) {
        // cuerpo
    },
    $arg3
);
```

2.7 Conclusión

Hay muchos elementos de estilo y práctica que se han omitido en esta guía. Éstos incluyen pero no están limitados a:

- Declaración de variables globales y constantes globales
- Declaración de funciones
- Operadores y asignaciones
- Alineación entre líneas
- Bloques de comentarios y documentación
- Prefijos y sufijos de nombres de clase
- Mejores prácticas

Futuras recomendaciones PUEDEN revisar y ampliar esta guía para abordar estos u otros elementos de estilo y práctica.