PAGINADOR 3.0.0

La utilidad de esta rutina es el paginado de un conjunto de datos, con total independencia de donde provengan; es decir no importa que sea un arreglo un resultado de base de datos o un conjunto de archivos. De lo único que necesita tener conocimiento es la cantidad de datos a paginar, cuantos datos se muestran por página y cuantos enlaces del paginador se quieren mostrar. Primero hagamos una vista de los métodos que nos proporciona la clase.

Métodos:

`__construct(integer $crpp, integer $cep) : void`

$crpp: es la cantidad de datos que se mostraran en cada página, por ejemplo productos, noticias, textos etc. (Por defecto 10).

$cep: cantidad de enlaces del navegador, si la paginación es muy larga la podemos acotar al número que queramos a través de este parámetro. (Por defecto 10).

`getCantidadPaginas() : integer`

Nos retorna el cálculo del total de páginas , útil para la presentación de datos como Viendo pagina 5 de 21.

`getHtmlPaginacion(string $varGet, string $cont ) : array`

$varGet: es la variable que se utiliza para pasar por url el enlace de paginación. Por defecto se utiliza como variable de propagación **página**.

$cont: tag html contenedor de la clase de estilo o sea propiedad class, por defecto se utiliza li.

Retorna un arreglo que contiene en cada elemento del mismo el código HTML para cada uno de los enlaces de la paginación.

```
<li class="next" ><a href="?pagina=1" title="Pagina Siguiente" >Siguiente >></a></li>
```

`getPaginacion() : array`

Retorna un arreglo bi-dimensional, cada elemento contiene las claves número, vista, title, class, off (en caso de existir). Numero representa el valor a pasar por la variable de propagación, vista es el valor que se mostrara (Existe un desfasaje de 1 entre número y vista), title es la propiedad title para poner un mensaje al posarse sobre el enlace y class el estilo que se quiere para el enlace, off en caso de existir indica que el enlace siempre se deberá mostrar y off es el estilo para ese caso.

`paginar(integer $pagina, integer $cantidadDeResultados) : array`

$pagina: es la página que se debe desplegar.

$cantidadDeResultados: es el total de datos.

Retorna el mismo arreglo que getPaginacion()

`setCantidadEnlaces(integer $cantidad) : void`

$cantidad: Cantidad de enlaces que se mostraran en la barra de navegación, no

incluyen los enlaces especiales como siguiente, anterior etc.

`setCantidadRegistros(integer $cantidad) : void`

$cantidad: es la cantidad de datos que se van a mostrar por cada página ej. Cantidad de productos a desplegar, cantidad de noticias etc.

`setClass(string $titulo, string $clase, string $claseOff) : void`

Configura que tipo de estilo se aplica a un determinado enlace.
Los tipos de enlace son diferenciados por los títulos, esta versión del paginador cuenta con 7 de ellos, estos son:

- **primero**: es el primer enlace del total de resultados (primera página).
- **bloqueAnterior**: dado a que nos podemos mover en bloque es decir si estamos viendo las páginas de la 21 a la 30 y cliqueamos en este enlace nos mostrara las páginas 11 a 20.
- **anterior**: es el tradicional enlace de ir una página atrás de la actual.
- **siguiente**: avanzamos una página.
- **bloqueSiguiente**: avanzamos un bloque.
- **ultimo**: es el último enlace del total de resultados (última página).
- **numero**: son los enlaces de navegación que corresponden a páginas comunes.
- **actual**: es la página que estamos viendo.

$titulo: es uno de los títulos mencionados anteriormente.
$clase: es el estilo que se le quiere dar a un determinado enlace.
Si se usa este método con getHtmlPaginacion entonces al tag contenedor se le añade este estilo Ej.: <li class="estilo"><a href=…….. . Si se usa con la forma tradicional simplemente es un valor más del arreglo retornado.
Tenemos tres posibilidades no configurar estilo es decir no usar el método con lo cual, no se usara el tag html contenedor en ese tipo de título.
Usar el método pasando un estilo, con lo cual el enlace quedara encerrado en el contenedor y con estilo ej.: <tagContenedor class='estilo'>anterior</tagContenedor>
Si solo se quiere que el enlace quede entre los Tag contenedores si ningún estilo lo que hacemos es enviar el string '<>', en este caso o sea setClass('numero', '<>').
$claseOff: se usa para que si un enlace no va a ser mostrado si le configuramos estilo lo obligamos a aparecer. Es el caso por ejemplo de anterior o primero que a veces no es necesario pero si siempre lo queremos le ponemos un estilo.

`setMarcador(string $antes, string $despues) : void`

Establece entre que marcadores se mostrara la página actual para diferenciarla del resto, por defecto es | (barra vertical).
Ej.: 1 2 3 4 |5| 6 7 ……
$antes: marcador a mostrar antes del número de página.
$*despues:* marcador a mostrar después del número de página.

`setOmitir(array $omitir) : void`

Establece que tipos de enlaces no serán mostrados en la barra de navegación, útil para cuando solo se desea hacer una navegación anterior, siguiente o eliminar bloque anterior y siguiente o no mostrar la página actual.

$omitir: es un arreglo donde cada elemento contiene uno de los títulos vistos en setClass que no se quiere mostrar.

Ej.: $paginador->setOmitir(array('primero', 'bloqueAnterior','bloqueSiguiente', 'actual', 'numero')); con esto logramos solo los enlaces anterior y siguiente.

`setPropagar(Array $variables) : void`

Se encarga de incluir en los enlaces de navegación las variables que se necesitan propagar, si se usa el modo tradicional no es necesario, aparece en esta versión complementando al método getHtmlPaginacion() dado que llega el enlace ya confeccionado al usuario.

$variables: arreglo de las variables que se desea propagar es del tipo arreglo usado en setOmitir. Las variables serán propagadas por url por lo cual se deberán levantar luego a través del método $_GET.

`setTitulosTitle(string $titulo, string $valor) : void`

Configura la propiedad title del enlace, es el toltip que aparece cuando nos paramos sobre un vínculo.

$titulo: es uno de los valores que se mencionaron en el método setClass.

$valor: es el comentario que queremos que se despliegue al pasar por arriba del enlace.

`setTitulosVista(string $titulo, string $valor) : void`

Establece que se mostrara en los enlaces, en el caso de los numéricos si se configura con este método se agregara adelante del numero el comentario pasado.

$titulo: es uno de los valores que se mencionaron en el método setClass.

$valor: es el comentario que queremos mostrar como vinculo.

`setUrlDestino(string $url) : void`

Configura la url destino del paginador.

$url: dirección web destino del enlace de la barra de navegación.

Esos son los métodos con que contamos para realizar nuestras paginaciones, a modo de ejemplo y con el método nuevo getHtmlPaginacion hemos logrado los siguientes estilos.

Los estilos fueron extraídos de dos sitios distintos y adaptados a la rutina:

- http://mis-algoritmos.com/some-styles-for-your-pagination
- http://blog.timersys.com/php/paginacion-con-php-y-mysql-3-estilos/

Vamos a explicar el funcionamiento de un ejemplo Básico:

```php
<?php
// Parametros a ser usados por el Paginador.
$cantidadRegistrosPorPagina = 10;
$cantidadEnlaces            = 10; // Cantidad de enlaces que tendra el paginador.
$totalRegistros             = 1653;
$pagina                     = isset($_GET['pagina'])? $_GET['pagina'] : 0;

```

La rutina requiere de los datos siguientes para poder funcionar correctamente. La cantidad de registros por página, por ejemplo si tenemos un carrito de compra y por cada búsqueda desplegamos 20 productos entonces este valor será 20. Es importante la coincidencia entre la cantidad de datos y este número pues si no es así la paginación dará resultados incorrectos. Cantidad de enlaces, en este caso es cuantos vínculos (paginas) presentara la rutina en la barra de navegación sin contar los especiales como ser anterior, siguiente, primero, etc. Total de registros, este puede ser el total de la consulta mysql, si fuera un arreglo el valor extraído del método count, es decir el total de los datos que se necesita paginar y por último la página en que nos encontramos, generalmente proviene de la url o si es nuestra primera llegada a la rutina es 0 (cero). Con estos parámetros ya podemos tener una configuración básica.

```php
// Comenzamos incluyendo el Paginador.
require_once 'Paginador.php';

// Instanciamos la clase Paginador
$paginador          = new Paginador();

```

Bueno aquí, algo común incluimos la rutina y la instanciamos para tener el objeto a disposición. Ya en el constructor podemos enviar la cantidad de registro por página y la

cantidad de enlaces de la barra de navegación. Al no pasarse asume 10 para cada uno de los valores.

```
22      // Configuramos cuanto registros por pagina que debe ser igual a el limit de la consulta mysql
23      $paginador->setCantidadRegistros($cantidadRegistrosPorPagina);
24      // Cantidad de enlaces del paginador sin contar los no numericos.
25      $paginador->setCantidadEnlaces($cantidadEnlaces);
26
```

En caso de no haber configurado al crear el objeto los parámetros antes mencionados, podemos usar los métodos set correspondientes. Ya con estos datos tenemos una paginación básica. Así que ya podríamos enviar a generar nuestra barra de navegación.

```
30      // Y mandamos a paginar desde la pagina actual y le pasamos tambien el total
31      // de registros de la consulta mysql.
32      $datos              = $paginador->paginar($pagina, $totalRegistros);
33
```

Con este último código enviamos a paginar, no es necesario recoger el resultado del método paginar en este momento, aunque conveniente si vamos a realizar una paginación con el método tradicional (Ver post Anteriores), igual la clase nos ofrece un método para recuperar ese arreglo en caso de requerirlo (getPaginacion).

```
36      $enlaces = $paginador->getHtmlPaginacion('pagina');
37      ?>
38      <div class="badoo">
39      <?php
40          foreach ($enlaces as $enlace) {
41              echo $enlace . "\n";
42          }
43      ?>
44      </div><br/><br />
```

Para este pequeño manual elegimos la nueva funcionalidad, la que envía un arreglo con cada enlace pronto para ser impreso. Para ello usamos el método getHtmlPaginacion y le pasamos la variable que queremos que propague el vínculo de página y por ahora nada más dado que queremos la paginación más básica de todas. Y la imagen siguiente muestra lo que obtuvimos. Claro que utiliza la hoja de estilos badoo, pero no se necesitó más configuración.

Ir a la pagina 129

ALGUNAS CONFIGURACIONES AVANZADAS O DE DISEÑO.

Si deseamos saber cuál es el total de páginas, para dar mayor información al usuario. Podemos usar el método getCantidadPaginas de la siguiente forma

```
44      <br /><br />Viendo la pagina :
45      <?php
46          echo ($pagina + 1) . ' de ' . $paginador->getCantidadPaginas();
47      ?>
48      </div>
```

y obtendríamos el siguiente paginador, dando información del total de páginas que existen.

Viendo la pagina : 132 de 166

Nota: antes de seguir con las opciones de diseño, vale aclarar que es conveniente que estas se encuentren entre la instancia de la clase y la llamada al método metodoPaginar.

Propagación De Variables: Dado que el enlace ya viene pronto para visualizar, se agregó el método setPropagar, el cual envía variables por url y deben ser levantados con el método GET, no era necesario en el modo tradicional.

Supongamos que queremos propagar las variables categoría y subCategoria de una familia de productos para ello debemos ingresar la siguiente línea de código.

```
28      // Propagar Variables
29      $paginador->setPropagar(array('categoria', 'subCategoria'));
30
```

Muchas veces mostramos una galería de imágenes o tipo de datos que solo requieren un enlace **anterior – siguiente**. Para ello contamos con el método setOmitir, y unas configuraciones extras.

```
32      // Primero le decimos que solo queremos mostrar un registro por pagina
33      $paginador->setCantidadRegistros(1);
34
35      // Luego eliminamos los enlaces que no necesitamos
36      $paginador->setOmitir(array('primero',
37                                  'bloqueAnterior',
38                                  'bloqueSiguiente',
39                                  'ultimo',
40                                  'actual',
41                                  'numero'));
42
```

Primero que nada configuramos la cantidad de registros por página, para decirle a la rutina que se desplegara un registro por vez, luego le pasamos un arreglo diciendo que títulos no queremos mostrar. Tenemos que tener cuidado pues si no omitimos número y la cantidad de enlace no la modificamos a 1 también nos mostrara los enlaces configurados o 10 en su defecto. Si quisiéramos mostrar aparte de siguiente y anterior el número actual en medio de los enlaces entonces en el arreglo no incluiríamos el valor actual.

< >

Viendo la pagina : 118 de 1653

Esta imagen representa la primera situación expuesta. La segunda simplemente contiene el número de la página que se está visitando en medio de los enlaces.

Ahora vemos que no nos convence las flechas de anterior y siguiente, y queremos los textos. Para lograr eso usamos el método setTitulosVista y le pasamos los textos deseados.

```
43      // Configuramos las vistas
44      $paginador->setTitulosVista('anterior', '<< anterior');
45      $paginador->setTitulosVista('siguiente', 'siguiente >>');
46
```

<< anterior siguiente >>

Viendo la pagina : 124 de 1653

También contamos con el método setTitulosTitle que nos permite cambiar el toltip que se despliega al pasar el ratón sobre un enlace, no pondremos un ejemplo dado que es idéntico el tratamiento que al método anterior.

Ahora un ejemplo de configuración de estilos, la configuración de los mismos se realiza con el método setClass. Para comenzar volvamos a un ejemplo básico como el siguiente.

```
<body>
    <?php
    error_reporting(E_ALL | E_STRICT);
    $cantidadRegistrosPorPagina = 10;
    $cantidadEnlaces            = 10;
    $totalRegistros             = 1653;
    $pagina                     = isset($_GET['pagina'])? $_GET['pagina'] : 0;

    require_once 'Paginador.php';
    $paginador  = new Paginador();

    $paginador->setCantidadRegistros($cantidadRegistrosPorPagina);
    $paginador->setCantidadEnlaces($cantidadEnlaces);
    /** AQUI INCLUIREMOS NUESTRO CODIGO DE CONFIGURACION DE ESTILOS */
    $datos      = $paginador->paginar($pagina, $totalRegistros);
    $enlaces    = $paginador->getHtmlPaginacion('pagina');
    ?>
    <div class="pagination-digg">
    <?php
        foreach ($enlaces as $enlace) {
            echo $enlace . "\n";
        }
    ?>
    <br /><br />Viendo la pagina :
    <?php
        echo ($pagina + 1) . ' de ' . $paginador->getCantidadPaginas();
    ?>
    </div>

    <br/><br />
</body>
```

En este caso el ejemplo lo vamos a hacer con la css digg y el paginador solo con estos datos estaría quedando de la siguiente manera.

| Primero ... << < 15 16 17 18 19 |20| 21 22 23 24 > >> ... Ultimo |

Viendo la pagina : 20 de 166

Nosotros queremos obtener:


```
<ul id="pagination-digg">
        <li class="previous-off">«Previous</li>
        <li class="active">1</li>
        <li><a href="?page=2">2</a></li>
        <li><a href="?page=3">3</a></li>
        <li><a href="?page=4">4</a></li>
        <li><a href="?page=5">5</a></li>
        <li><a href="?page=6">6</a></li>
        <li><a href="?page=7">7</a></li>
        <li class="next"><a href="?page=8">Next »</a></li>
</ul>
```

Lo primero que resalta es que se utiliza una lista sin orden (<ul>) para encerrar el conjunto de enlaces, dado que esto queda fuera del alcance del paginador puesto que no pertenece a ningún enlace sino a todos, lo tendremos que incluir fuera de nuestro bucle de vínculos.

```
<ul id="pagination-digg"> // Linea 25
</ul> // Linea 35
```

O sea cambiamos nuestro div por ul y el correspondiente div de cierre por el ul de cierre aparte del atributo class por id. Luego podemos notar que cada enlace esta contenido entre los tags , en este caso no es necesario hacer nada más pues la rutina por defecto utiliza li, pero si nos quisiéramos asegurar igualmente el contenedor HTML li hacemos lo siguiente:

```
$enlaces=$paginador->getHtmlPaginacion('pagina', 'li'); // en la página 23
```

Hasta aquí nuestro código esta de la siguiente manera, vamos a sacar viendo página x de n, pues no es necesario para el ejemplo y queda más claro sin ello.

```
42    $enlaces    = $paginador->getHtmlPaginacion('pagina', 'li');
43    ?>
44    <ul id="pagination-digg">
45    <?php
46        foreach ($enlaces as $enlace) {
47            echo $enlace . "\n";
48        }
49    ?>
50    </ul>
```

Si analizamos el archivo digg.css vemos que tenemos los estilos li, a, previous-off, next-off, next, previous, active, link, visited, hover. Por lo que podemos inferir es que:

- previous-off: Seria para mostrar anterior pero cuando no está activo el enlace.
- Next-off: para mostrar siguiente con el enlace sin activar (no disponible).
- Next: siguiente común.
- Previous: anterior común.
- Active: enlace en el que estamos parados.
- El resto son estilo sobre tags Html.

El último método que veremos es el de configuración de los estilos setClass. Este método nos permite tener un estilo particular para cada uno de los títulos que maneja la rutina (anterior, siguiente, bloqueAnterior….., ultimo, etc). Veamos que código nos genera el paginador hasta el momento:

```
<ul id="pagination-digg">
|1|
<a href="?pagina=1" title="Ir a la pagina 2" >2</a>
<a href="?pagina=2" title="Ir a la pagina 3" >3</a>
<a href="?pagina=3" title="Ir a la pagina 4" >4</a>
<a href="?pagina=4" title="Ir a la pagina 5" >5</a>
<a href="?pagina=5" title="Ir a la pagina 6" >6</a>
<a href="?pagina=6" title="Ir a la pagina 7" >7</a>
<a href="?pagina=7" title="Ir a la pagina 8" >8</a>
<a href="?pagina=8" title="Ir a la pagina 9" >9</a>
<a href="?pagina=9" title="Ir a la pagina 10" >10</a>
<a href="?pagina=1" title="Pagina Siguiente" >></a>
<a href="?pagina=15" title="Bloque Siguiente" >>></a>
<a href="?pagina=236" title="Ir a la Ultima Pagina" >... Ultimo |</a>
</ul>
```

Tenemos bastantes diferencias, en el digg.css tomado como ejemplo, todas las páginas están contenidas dentro del tag li, algunos li contienen clases y otros no. Para intentar acomodar esto tenemos como dijimos el método setClass, que nos permite configurar los estilos de cada tipo de título de nuestro paginador.

```
setClass(string $titulo, string $clase, string $claseOff) : void
```

Título: es como sabemos el nombre para el enlace que queremos dar estilo.

Clase: nos da 3 posibilidades de aplicar estilos,

1> no poner nada, situación por defecto y en este caso el enlace va sin ser encerrado entre los tag contenedores, es como tenemos ahora el código.
2> Poner <>, y en este caso no se usan estilos pero el enlace se encierra entre los tag contenedores.
3> Poner un estilo y aquí se pondrá el tag contenedor y se le pondrá el atributo class con el estilo seleccionado.

ClaseOff: la rutina por defecto asume que si es un enlace que ya se tiene alcance a través de un número, ese enlace no se presenta. Es decir si nosotros estamos por ejemplo en el enlace 2 o 3 donde todavía vemos el enlace 1 a la vista, entonces la rutina no nos mostrara ir a la primera página, puesto a que tenemos el enlace directo. Si queremos que este enlace por estética este siempre presente entonces ponemos un estilo tipo deshabilitado y dicho enlace será mostrado siempre y en caso de no ser necesario se le aplica el estilo deshabilitado.

Como llegamos al estilo deseado, primero vemos que el estilo siempre nos muestra 7 enlaces.

```
$paginador->setCantidadEnlaces(7);
```

Observemos que nunca pone última, primera y tampoco tiene los enlaces exclusivos bloque anterior y bloque siguiente, entonces los deshabilitaremos.

```
$paginador->setOmitir(array('primero', 'bloqueAnterior', 'ultimo', 'bloqueSiguiente'));
```

La página actual no está enmarcada por ningún carácter.

```
$paginador->setMarcador(null, null);
```

Los enlaces anterior y siguiente no son simples flechas,

```
$paginador->setTitulosVista('anterior',  '<<Previous');
$paginador->setTitulosVista('siguiente', 'next>>');
```

El enlace anterior o previo, utiliza el estilo previous si está activo y si no previous-off.

```
$paginador->setClass('anterior',        'previous',     'previous-off');
```

Para el enlace siguiente, lo mismo que anterior pero con next y next-off

```
$paginador->setClass('siguiente',       'next',         'next-off');
```

El enlace de la página actual se le adjudica el estilo active

```
$paginador->setClass('actual',          'active');
```

Y por último los enlaces comunes de página solo se le pone el contenedor sin estilos

```
$paginador->setClass('numero',          '<>');
```

Bueno con eso ya obtenemos nuestra barra de navegación.

Y aquí el código de personalizacion.

```
30    $paginador->setCantidadEnlaces(7);
31    $paginador->setOmitir(array('primero',
32                                'bloqueAnterior',
33                                'ultimo',
34                                'bloqueSiguiente'));
35    $paginador->setMarcador(null, null);
36
37    $paginador->setTitulosVista('anterior',  '<<Previous');
38    $paginador->setTitulosVista('siguiente', 'next>>');
39
40    $paginador->setClass('anterior',        'previous',     'previous-off');
41    $paginador->setClass('siguiente',       'next',         'next-off');
42    $paginador->setClass('actual',          'active');
43    $paginador->setClass('numero',          '<>');
44
45    $datos      = $paginador->paginar($pagina, $totalRegistros);
46    $enlaces    = $paginador->getHtmlPaginacion('pagina', 'li');
```

Bueno espero que sea de utilidad y que hayan comprendido todos los pasos.