View

View is the class responsible for rendering PHP views (in MVC parlance).

Namespace: `\`
File location: `lib/base.php`

What is a view?

A view is a representation of your project data. It is considered as a best practice to separate the code generating the data from the code generating the representation.

For example, let's consider how you could generate a website sitemap in traditional PHP:

```
// first we generate the list of URLs (Web pages)
$urls=array('http://domain.tld/home','http://domain.tld/contact','http://domain.tld/a
bout');

// at the end of the code, we display the list as an HTML view:
header('Content-Type: text/html; charset=UTF-8');
echo "<html><table><tr><td>$urls[0]</td></tr><tr><td>$urls[1]</td></tr><tr><td>$urls
[2]</td></tr></table></html>";

// or as a XML view:
header('Content-Type: text/xml; charset=UTF-8');
echo "<?xml><urlset><url><loc>$urls[0]</loc></url><url><loc>$urls[1]</loc></url><url>
<loc>$urls[2]</loc></url></urlset>";

// or as a CSV view:
header('Content-Type: text/csv; charset=UTF-8');
echo implode("\r\n",$urls);
```

Render a view

The View class makes it easy to separate the two steps defined above, since it requires rendering a filename and a data hive:

```
string render ( string $file [, string $mime = 'text/html' [, array $hive = NULL, [ i
nt $ttl = 0 ]]] )
```

Note:
The `$mime` argument is responsible for generating the proper auto-generated `Content-type` header.
If no data `$hive` is provided, the global F3 hive is used.

Here's how to use it in your route handler using the F3 hive:

```
$f3->set('urls',
    array(
        'http://domain.tld/home',
        'http://domain.tld/contact',
        'http://domain.tld/about'
    )
);
$view=\View::instance();
echo $view->render('myview.html','text/html');
// or
echo $view->render('myview.xml','text/xml');
// or
echo $view->render('myview.csv','text/csv');
```

If you prefer to pass only the relevant variables to your view:

```
$urls=array('http://domain.tld/home','http://domain.tld/contact','http://domain.tld/a
bout');
$view=\View::instance();
echo $view->render('myview.html','text/html',array('urls'=>$urls));
// or
echo $view->render('myview.xml','text/xml',array('urls'=>$urls));
// or
echo $view->render('myview.csv','text/csv',array('urls'=>$urls));
```

myview.html

```
<html>
<head>
...
</head>
<body>
<table>
<?php foreach($urls as $url):?>
  <tr>
    <td>
      <?php echo $url;?>
    </td>
  </tr>
<?php endforeach;?>
</body>
</html>
```

myview.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<?php foreach($urls as $url):?>
  <url>
    <loc>
      <?php echo $url;?>
    </loc>
  </url>
<?php endforeach;?>
</urlset>
```

myview.csv

```
<?php foreach($urls as $url):?>
<?php echo $url;?>
<?php endforeach;?>
```

Methods

afterrender

call a function after view is rendered, before it is written to `/tmp` storage

```
string afterrender ( string|callback $func )
```

Call the function specified in `$func` .

When function is called, the rendered view is passed to the function as a string. The called function needs to return the modified rendered view after any modification are done from within the callback function. You can call this hook multiple times to assign more than one function if needed. Utilizes Base->call (base#call) to trigger the function and therefore accepts a callback string as well.

esc

Encode characters to equivalent HTML entities

```
string esc ( mixed $arg )
```

Usage:

```
echo $view->esc("99 bottles of <b>beer</b> on the wall. <script>alert(1);</script>");
// 99 bottles of &amp;lt;b&amp;gt;beer&amp;lt;/b&amp;gt; on the wall. &amp;lt;script&
amp;gt;alert(1);&amp;lt;/script&amp;gt;
```

This also works with arrays and object properties:

```
$myArray = array('<b>foo</b>',array('<script>alert(1)</script>'),'key'=>'<i>foo</i
>');
print_r($view->esc($myArray));
/*
    [0] => &amp;lt;b&amp;gt;foo&amp;lt;/b&amp;gt;
    [1] => Array
        (
            [0] => &amp;lt;script&amp;gt;alert(1)&amp;lt;/script&amp;gt;
        )
    [key] => &amp;lt;i&amp;gt;foo&amp;lt;/i&amp;gt;
*/

$myObj = new stdClass();
$myObj->title = '<h1>Hello World</h1>';
var_dump($view->esc($myObj));
/*
    object(stdClass)#23 (1) {
      ["title"] => string(32) "&amp;lt;h1&amp;gt;Hello World&amp;lt;/h1&amp;gt;"
    }
*/
```

If the system var **ESCAPE** is turned on (it is by default), then every hive key access using a template token like **{{@myContent}}** automatically get escaped by this function. If this is not desired, look into the template (template) section for further post processors.

raw

Decode HTML entities to equivalent characters

```
string raw ( mixed $arg )
```

Example:

```
$view->raw("99 bottles of &amp;lt;b&amp;gt;beer&amp;lt;/b&amp;gt; on the wall. &amp;l
t;script&amp;gt;alert(1);&amp;lt;/script&amp;gt;");
// 99 bottles of <b>beer</b> on the wall. <script>alert(1);</script>
```

This method also handles nested array elements and objects properties, like `$view->esc()` does too.