EFTEC / **BladeOne**

The standalone version Blade Template Engine without Laravel in a single php file and without dependencies
http://www.southprojects.com/

#template-engine #blade-template #php #standalone #laravel #loop #view

100 commits | **2** branches | **9** releases | **4** contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

ProfessorJoe Added BladeOneLang		Latest commit 7a91757 on Dec 19, 2017
examples	Added BladeOneLang	2 months ago
BladeOne.php	2.2 **@asset**	2 months ago
BladeOneCache.md	July the 1st	2 years ago
BladeOneCache.php	Version 1.4	2 years ago
BladeOneCacheRedis.php	Added BladeOneLang	2 months ago
BladeOneCustom.php	Added an example of a custom component	7 months ago
BladeOneHtml.md	2017-02-20	a year ago
BladeOneHtml.php	1-9	5 months ago
BladeOneHtmlBootstrap.php	1-9	5 months ago
BladeOneLang.md	Added BladeOneLang	2 months ago
BladeOneLang.php	Added BladeOneLang	2 months ago
BladeOneLogic.md	Version 1.3	2 years ago
BladeOneLogic.php	Version 1.4	2 years ago
LICENSE	change of file	2 years ago
README.md	Added BladeOneLang	2 months ago
composer.json	2.2 **@asset**	2 months ago
readme.template.md	2016-09-24	a year ago

README.md

BladeOne
The root of new Projects

BladeOne

BladeOne is a standalone version of Blade Template Engine that uses a single php file and can be ported and used in different projects.

license MIT | maintained stale (as of 2018) | npm >4.1 | php >5.4 | php 7.x | docs 40%

Introduction (From Laravel webpage)

Blade is the simple, yet powerful templating engine provided with Laravel. Unlike other popular PHP templating engines, Blade does not restrict you from using plain PHP code in your views. All Blade views are compiled into plain PHP code and cached until they are modified, meaning Blade adds essentially zero overhead to your application. Blade view files use the .blade.php file extension and are typically stored in the resources/views directory.

About this version

By standard, Blade is part of Laravel (Illuminate components) and for to use it, you requires to install Laravel and Illuminate-view components. Blade as a template engine is pretty nice and clear. Also it generates a (some that) clean code. And its starting to be considered a de-facto template system for php (Smarty has been riding off the sunset since years ago). So, if we are able to use it without Laravel then its a big plus for many projects. In fact, in theory its is even possible to use with Laravel. Exists different version of Blade Template that runs without Laravel but most requires 50 or more files and those templates add a new level of complexity:

- More files to manages.
- Changes to the current project (if you want to integrate the template into an existent one)
- Incompatibilities amongst other projects.
- Slowness (if your server is not using op-cache)
- Most of the code in the original Blade is used for future use, including the chance to use a different template engine.
- Some Laravel legacy code.

This project uses a single file called BladeOne.php and a single class (called BladeOne). If you want to use it then include it, creates the folders and that's it!. Nothing more (not even namespaces)*[]:

Why to use it instead of native PHP?

Separation of concerns

Let's say that we have the next code

```
<?php
//some php code
// some html code
// more php code
// more html code.
?>
```

It leads to a mess of a code. For example, let's say that we oversee changing the visual layout of the page. In this case, we should change all the code and we could even break part of the programming.
Instead, using a template system works in the next way:

```
<?php
// some php code
ShowTemplate();
?>
```

We are separating the visual layer from the code layer. As a plus, we could assign a non-php-programmer in charge to edit the template, and he/she doesn't need to touch or know our php code.

Security

Let's say that we have the next exercise (it's a dummy example)

```
<?php
$name=@$_GET['name'];
Echo "my name is ".$name;
?>
```

It could be separates as two files:

```
<?php // index.php
$name=@$_GET['name'];
Include "template.php"
?>
```

```
<?php // template.php
Echo “my name is “.$name;
?>
```

Even for this simple example, there is a risk of hacking. How? A user could sends malicious code by using the GET variable, such as html or even javascript. The second file should be written as follow:

```
<?php // template.php
Echo “my name is “.html_entities($name);
?>
```

html_entities should be used in every single part of the visual layer (html) where the user could injects malicious code, and it’s a real tedious work. BladeOne does it automatically.

```
// template.blade.php
My name is {{$name}}
```

Easy to use

BladeOne is focused in templates and the syntax of the code is aiming to be clean.

Let's consider the next template:

```
<select>
    <? foreach($countries as $c) { ?>
        <option value=<? echo html_entities($c->value); ?> > <? echo html_entities($c->text); ?></option>
    <? } ?>
</select>
```

With BladeOne, we could do the same with

```
<select>
    @foreach($countries as $c)
        <option value={{$c->value}} >{{echo html_entities($c->text)}}</option>
    @nextforeach
</select>
```

And with html extension we could even reduce to

```
@select('id1')
    @items($countries,'value','text','','')
@endselect()
```

Performance

This library works in two stages.

The first is when the template is called the first time. In this case, the template is compiled and stored in a folder.
The second time the template is called then, it uses the compiled file. The compiled file consist mainly in native PHP, so **the performance is equals than native code.** since the compiled version IS PHP.

Scalable

You could add and use your own function by adding a new method (or extending) to the bladeone class. NOTE: The function should starts with the name "compile"

```
protected function compileMyFunction($expression)
{
    return $this->phpTag . "echo 'YAY MY FUNCTION IS WORKING'; ?>";
}
```

Where the function could be used in a template as follow

```
@myFunction('param','param2'...)
```

Alternatively, BladeOne allows to run arbitrary code from any class or method if its defined.

```
{{SomeClass::SomeMethod('param','param2'...)}}
```

Usage

example.php:

```
<?php
include "BladeOne.php";
Use eftec\bladeone;

$views = __DIR__ . '/views';
$cache = __DIR__ . '/cache';
define("BLADEONE_MODE",1); // (optional) 1=forced (test),2=run fast (production), 0=automatic, default value.
$blade=new bladeone\BladeOne($views,$cache);
echo $blade->run("hello",array("variable1"=>"value1"));
```

Or using composer's autoload.php

```
<?php
require "vendor/autoload.php";

Use eftec\bladeone;

$views = __DIR__ . '/views';
$cache = __DIR__ . '/cache';
define("BLADEONE_MODE",1); // (optional) 1=forced (test),2=run fast (production), 0=automatic, default value.
$blade=new bladeone\BladeOne($views,$cache);
echo $blade->run("hello",array("variable1"=>"value1"));
```

(modify composer.json as follow) and run "composer update"

```
"autoload": {
  "psr-4": {
    "eftec\\": "vendor/eftec/"
  }
}
```

Where $views is the folder where the views (templates not compiled) will be stored. $cache is the folder where the compiled files will be stored.

In this example, the BladeOne opens the template **hello**. So in the views folders it should exists a file called **hello.blade.php**

views/hello.blade.php:

```
<h1>Title</h1>
{{$variable1}}
```

Business Logic/Controller methods

constructor

```
$blade=new bladeone\BladeOne($views,$cache);
```

- BladeOne(templatefolder,compiledfolder) Creates the instance of BladeOne.

- templatefolders indicates the folder (without ending backslash) of where the template files (*.blade.php) are located.
- compiledfolder indicates the folder where the result of files will be saves. This folder should has write permission. Also, this folder could be located outside of the Web Root.

run

```
echo $blade->run("hello",array("variable1"=>"value1"));
```

- run([template,[array]) Runs the template and generates a compiled version (if its required), then it shows the result.
- template is the template to open. The dots are used for to separate folders. If the template is called "folder.example" then the engine tries to open the file "folder\example.blade.php"
- array (optional). Indicates the values to use for the template. For example ['v1'=>10'], indicates the variable $v1 is equals to 10

BLADEONE_MODE (global constant) (optional)

```
define("BLADEONE_MODE",1); // (optional) 1=forced (test),2=run fast (production), 0=automatic, default value.
```

- BLADEONE_MODE Is a global constant that defines the behaviour of the engine.
- 1=forced. Indicates that the engine always will compile the template.
- 2=fast. Indicates that the engine always will use the compiled version

Template tags

Template Inheritance

In the master page (layout)

Tag	Note	status
@section('sidebar')	Start a new section	0.2b ok
@show	Indicates where the content of section will be displayed	0.2 ok
@yield('title')	Show here the content of a section	0.2b ok

Using the master page (using the layout)

Tag	Note	status
@extends('layouts.master')	Indicates the layout to use	0.2b ok
@section('title', 'Page Title')	Sends a single text to a section	0.2b ok
@section('sidebar')	Start a block of code to send to a section	0.2b ok
@endsection	End a block of code	0.2b ok
@parent	Show the original code of the section	REMOVED(*)

Note :(*) This feature is in the original documentation but its not implemented neither its required. May be its an obsolete feature.

variables

Tag	Note	status
{{$variable1}}	show the value of the variable using htmlentities (avoid xss attacks)	0.2b ok

Tag	Note	status
@{{$variable1}}	show the value of the content directly (not evaluated, useful for js)	0.2b ok
{!!$variable1!!}	show the value of the variable without htmlentities (no escaped)	0.2b ok
{{ $name or 'Default' }}	value or default	0.2b ok
{{Class::StaticFunction($variable)}}	call and show a function (the function should return a value)	0.2b ok

logic

Tag	Note	status
@if (boolean)	if logic-conditional	0.2b ok
@elseif (boolean)	else if logic-conditional	0.2b ok
@else	else logic	0.2b ok
@endif	end if logic	0.2b ok
@unless(boolean)	execute block of code is boolean is false	0.2b ok

loop

@for($variable;$condition;$increment) / @endfor

Generates a loop until the condition is meet and the variable is incremented for each loop

Tag	Note	Example
$variable	is a variable that should be initialized.	$i=0
$condition	is the condition that must be true, otherwise the cycle will end.	$i<10
$increment	is how the variable is incremented in each loop.	$i++

Example:

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}<br>
@endfor
```

Returns:

```
The current value is 0
The current value is 1
The current value is 2
The current value is 3
The current value is 4
The current value is 5
The current value is 6
The current value is 7
The current value is 8
The current value is 9
```

@foreach($array as $alias) / @endforeach

Generates a loop for each values of the variable.

Tag	Note	Example
$array	Is an array with values.	$countries
$alias	is a new variable that it stores each interaction of the cycle.	$country

Tag Note Example

Example: ($users is an array of objects)

```
@foreach($users as $user)
    This is user {{ $user->id }}
@endforeach
```

Returns:

```
This is user 1
This is user 2
```

@forelse($array as $alias) / @empty / @endforelse

Its the same than foreach but jumps to the @empty tag if the array is null or empty

Tag	Note	Example
$array	Is an array with values.	$countries
$alias	is a new variable that it stores each interaction of the cycle.	$country

Example: ($users is an array of objects)

```
@forelse($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse
```

Returns:

```
John Doe
Anna Smith
```

@while($condition) / @endwhile

Loops until the condition is not meet.

Tag	Note	Example
$condition	The cycle loops until the condition is false.	$counter<10

Example: ($users is an array of objects)

```
@set($whilecounter=0)
@while($whilecounter<3)
    @set($whilecounter)
    I'm looping forever.<br>
@endwhile
```

Returns:

```
I'm looping forever.
I'm looping forever.
I'm looping forever.
```

@splitforeach($nElem,$textbetween,$textend="") inside @foreach

This functions show a text inside a @foreach cycle every "n" of elements. This function could be used when you want to add columns to a list of elements.
NOTE: The $textbetween is not displayed if its the last element of the last. With the last element it show the variable $textend
''

Tag	Note	Example

Tag	Note	Example
$nElem	Number of elements	2, for every 2 element the text is displayed
$textbetween	Text to show	'
$textend	Text to show	'

Example: ($users is an array of objects)

```
<table border="1">
<tr>
@foreach($drinks7 as $drink)
    <td>{{$drink}}</td>
    @splitforeach(2,'</tr><tr>','</tr>')
    @endforeach
</table>
```

Returns a table with 2 columns.

@continue / @break

Continue jump to the next iteraction of a cycle. @Break jump out of a cycle.

Tag	Note	Example

Example: ($users is an array of objects)

```
@foreach($users as $user)
    @if($user->type == 1) // ignores the first user John Smith
    @continue
    @endif
    <li>{{ $user->type }} - {{ $user->name }}</li>

    @if($user->number == 5) // ends the cycle.
        @break
    @endif
@endforeach
```

Returns:

```
2 - Anna Smith
```

Sub Views

Tag	Note	status
@include('folder.template')	Include a template	0.2b ok
@include('folder.template',['some' => 'data'])	Include a template with new variables	0.2b ok
@each('view.name', $array, 'variable')	Includes a template for each element of the array	0.2b ok
Note: Templates called folder.template is equals to folder/template		

Comments

Tag	Note	status
{{-- Text --}}	Include a comment	0.2b ok

Stacks

Tag	Note	status
@push('elem')	Add the next block to the push stack	0.2b ok
@endpush	End the push block	0.2b ok
@stack('elem')	Show the stack	0.2b ok

@set (new for 1.5)

@set($variable=[value]) @set($variable) is equals to @set($variable=$variable+1)

- $variable define the variable to add. If not value is defined the it adds +1 to a variable.
- value (option) define the value to use.

Service Inject

Tag	Note	status
@inject('metrics', 'App\Services\MetricsService')	Used for insert a Laravel Service	NOT SUPPORTED

Extending Blade

Not compatible with the extension of Laravel's Blade.

component / slots

@asset (new for 2.2)

@asset('js/jquery.js')

Note: it requires to set the base address as

```
$obj=new BladeOne();
$obj->baseUrl="https://www.example.com/urlbase/";
```

Security: Don't use the variables $SERVER['HTTP_HOST'] or $SERVER['SERVER_NAME'] unless the server is protected or the address is sanitized.

Extensions Libraries (optional)

BladeOneHtml Documentation

BladeOneLogic Documentation

BladeOneCache Documentation

BladeOneLang Documentation

Definition of Blade Template

https://laravel.com/docs/5.2/blade

#Differences between Blade and BladeOne

- Laravel's extension removed.
- Dependencies to other class removed (around 30 classes).
- The engine is self contained.

- Setter and Getters removed. Instead, we are using the PHP style (public members).
- BladeOne doesn't support static calls.

Differences between Blade+Laravel and BladeOne+BladeOneHTML

Instead of use the Laravel functions, for example Form::select

```
{{Form::select('countryID', $arrayCountries,$countrySelected)}}
```

We have native tags as @select,@item,@items and @endselect

```
@select('countryID')
    @item('0','--Select a country--',$countrySelected)
    @items($arrayCountries,'id','name',$countrySelected)
@endselect()
```

This new syntaxis add an (optionally) a non-selected row. Also, BladeOneHTML adds multiple select, fixed values (without array), grouped select and many more.

#Version

- 2016-06-08 0.2. Beta First publish launch.
- 2016-06-09 1.0 Version. Most works. Added extensions and error control with a tag is not defined.
- 2016-06-09 1.1 Some fine tune.
- 2016-06-10 1.2 New changes. Added namespaces (for autocomplete and compatibility with composer)
- 2016-06-12 1.3 Lots of clean up. I removed some unused parameters. I fixed a problem with forced in BladeOne. I separates the doc per extension.
- 2016-06-24 1.4 Updates extensions. Now it uses strut instead of classes. Added a new extension BladeOneCache.
- 2016-07-03 1.5 New features such as **@set** command
- 2016-08-14 1.6 Some cleanups. Add new documentation
- 2017-02-20 1.6 More cleanups. Refactored file,image and other tags.
- 2017-04-09 1.8 Creates directory automatically. Some fixes. Add new feature **@splitforeach**.
- 2017-05-24 1.8 Maintenance. Now, it runs with or without mb_string module
- 2017-07-21 1.9 Components and Slots. Note: im not really convinced in its usability.
- 2017-09-28 2.0 Some fixes there and here.
  Fixed foreach bug when the name of the variable contains the letters 'as' for example @foreach($list**As**Fast as $v)
  Fixed @item (BladeOneHtml). Now, it considers null and 0 as different.
- 2017-10-20 2.1 Fixed with @parent
- 2017-12-14 2.2 Added @asset
- 2017-12-18 Added BladeOneLang

#Collaboration

You are welcome to use it, share it, ask for changes and whatever you want to. Just keeps the copyright notice in the file.

#Future I checked the code of BladeOne and i know that there are a lot of room for improvement.

#License MIT License. BladeOne (c) 2016 Jorge Patricio Castro Castillo Blade (c) 2012 Laravel Team (This code is based and use the work of the team of Laravel.)