# Money Documentation

***Release 2.0***

**Mathias Verraes**

February 03, 2016

Contents

CHAPTER 1

# Why a Money library for PHP?

Also see http://blog.verraes.net/2011/04/fowler-money-pattern-in-php/

This is a PHP implementation of the Money pattern, as described in *[Fowler2002]* :

> A large proportion of the computers in this world manipulate money, so it's always puzzled me that money isn't actually a first class data type in any mainstream programming language. The lack of a type causes problems, the most obvious surrounding currencies. If all your calculations are done in a single currency, this isn't a huge problem, but once you involve multiple currencies you want to avoid adding your dollars to your yen without taking the currency differences into account. The more subtle problem is with rounding. Monetary calculations are often rounded to the smallest currency unit. When you do this it's easy to lose pennies (or your local equivalent) because of rounding errors.

## 1.1 The goal

Implement a reusable Money class in PHP, using all the best practices and taking care of all the subtle intricacies of handling money. I hope to add a lot more features, such as dealing with major units and subunits in currencies, currency conversion, string formatting and parsing, ... Other ideas include integration with Doctrine2, which should make it easier to store money in a database transparently.

CHAPTER 2

# Getting started

All amounts are represented in the smallest unit (eg. cents), so USD 5.00 is written as

```php
<?php
$fiver = new Money(500, new Currency('USD'));
// or shorter:
$fiver = Money::USD(500);
```

## 2.1 Autoloading

You'll need an autoloader. Money is PSR-0 compatible, so if you are using the Symfony2 autoloader, this will do:

```php
<?php
use Symfony\Component\ClassLoader\UniversalClassLoader;

$loader = new UniversalClassLoader;
$loader->registerNamespaces(array(
    'Money' => __DIR__ . '/vendor/money/lib/',
));
$loader->register();
```

CHAPTER 3

# Immutability

Jim and Hannah both want to buy a copy of book priced at EUR 25.

```
<?php
$jim_price = $hannah_price = Money::EUR(2500);
```

Jim has a coupon for EUR 5.

```
<?php
$coupon = Money::EUR(500);
$jim_price->subtract($coupon);
```

Because `$jim_price` and `$hannah_price` are the same object, you'd expect Hannah to now have the reduced price as well. To prevent this problem, Money objects are **immutable**. With the code above, both `$jim_price` and `$hannah_price` are still EUR 25:

```
<?php
$jim_price->equals($hannah_price); // true
```

The correct way of doing operations is:

```
<?php
$jim_price = $jim_price->subtract($coupon);
$jim_price->lessThan($hannah_price); // true
$jim_price->equals(Money::EUR(2000)); // true
```

CHAPTER 4

# Allocation

My company made a whopping profit of 5 cents, which has to be divided amongst myself (70%) and my investor (30%). Cents can't be divided, so I can't give 3.5 and 1.5 cents. If I round up, I get 4 cents, the investor gets 2, which means I need to conjure up an additional cent. Rounding down to 3 and 1 cent leaves me 1 cent. Apart from re-investing that cent in the company, the best solution is to keep handing out the remainder until all money is spent. In other words:

```php
<?php
$profit = Money::EUR(5);
list($my_cut, $investors_cut) = $profit->allocate(array(70, 30));
// $my_cut is 4 cents, $investors_cut is 1 cent

// The order is important:
list($investors_cut, $my_cut) = $profit->allocate(array(30, 70));
// $my_cut is 3 cents, $investors_cut is 2 cents
```

CHAPTER 5

# Inspiration

- https://github.com/RubyMoney/money
- http://css.dzone.com/books/practical-php-patterns/basic/practical-php-patterns-value
- http://www.codeproject.com/KB/recipes/MoneyTypeForCLR.aspx
- http://www.michaelbrumm.com/money.html
- http://stackoverflow.com/questions/1679292/proof-that-fowlers-money-allocation-algorithm-is-correct
- http://timeandmoney.sourceforge.net/
- https://github.com/lucamarrocco/timeandmoney/blob/master/lib/money.rb
- http://joda-money.sourceforge.net/
- http://en.wikipedia.org/wiki/Currency_pair
- https://github.com/RubyMoney/eu_central_bank
- http://en.wikipedia.org/wiki/ISO_4217

# Bibliography

[Fowler2002] Fowler, M., D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford, Patterns of Enterprise Application Architecture, Addison-Wesley, 2002. http://martinfowler.com/books.html#eaa