
Comodojo Foundation Documentation

Release 1.0.0

Marco Giovinazzi

Feb 10, 2019

Contents

1 Configuration handler	3
1.1 Using Configuration Class	3
2 Data Access	5
3 Events facilities	7
4 Logging facilities	9
5 Timing	11
6 Generic utilities	13
6.1 Array Operations	13
6.2 Uid generator	15
7 Data filtering and validation	17
7.1 Data filtering	17
7.2 Data validation	17

This package provides foundation modules for comodojo libs and frameworks.

CHAPTER 1

Configuration handler

Classes in \Comodojo\Foundation\Base are designed to provide easy management of configuration statements across every comodojo lib.

The \Comodojo\Foundation\Base\Configuration class provides methods to set, update and delete statements using dot notation.

Note: Dot notation, as implemented here, is a handy way to navigate a tree of statements. Considering a yaml tree like:

```
log:
    enable: true
    name: applog
    providers:
        local:
            type: StreamHandler
            level: debug
            stream: logs/extenderd.log
cache:
    enable: true
    providers:
        local:
            type: Filesystem
            cache_folder: cache
```

Statement “log.providers.local.type” points directly to “StreamHandler” value

1.1 Using Configuration Class

A base configuration object can be created using standard constructor or static Configuration::create method. Constructor accepts an optional array of parameters that will be pushed to the properties’ stack.

```
<?php
$params = ["this"=>"is", "a"=>["config", "statement"]];
```

(continues on next page)

(continued from previous page)

```
$configuration = new \Comodojo\Foundation\Base\Configuration($params)  
  
// or, alternatively:  
// $configuration = \Comodojo\Foundation\Base\Configuration::create($params)
```

Once created, a configuration object offers methods to manage statements:

- Configuration::set (): set (or update) a statement
- Configuration::get (): get value of statement
- Configuration::has (): check if statement is defined
- Configuration::delete (): remove a statement from stack
- Configuration::merge (): merge a package of statements into current stack

Usage example:

```
<?php  
  
$params = ["this"=>"is", "a"=>["config", "statement"]];  
  
$configuration = \Comodojo\Foundation\Base\Configuration::create($params);  
  
var_dump($configuration->get("a"));  
  
$configuration->set("that", "value");  
  
var_dump($configuration->get("that"));
```

Produces:

```
array(2) {  
    [0] =>  
        string(6) "config"  
    [1] =>  
        string(9) "statement"  
}  
  
string(5) "value"
```

CHAPTER 2

Data Access

CHAPTER 3

Events facilities

CHAPTER 4

Logging facilities

CHAPTER 5

Timing

CHAPTER 6

Generic utilities

6.1 Array Operations

6.1.1 ArrayOps::circularDiffKeys

Perform a circular diff between two arrays using keys.

This method is useful to compute the actual differences between two arrays.

Usage:

```
<?php

$left = [
    "ford" => "perfect",
    "marvin" => "android",
    "arthur" => "dent"
];

$right = [
    "marvin" => "android",
    "tricia" => "mcmillan"
];

var_dump(\Comodojo\Foundation\Utils\ArrayOps::circularDiffKeys($left, $right));
```

It returns:

```
array(3) {
[0] =>
array(2) {
    'ford' =>
    string(7) "perfect"
    'arthur' =>
    string(4) "dent"
}
[1] =>
array(1) {
    'marvin' =>
}
```

(continues on next page)

(continued from previous page)

```
    string(7) "android"
}
[2] =>
array(1) {
    'tricia' =>
    string(8) "mcmillan"
}
}
```

6.1.2 ArrayOps::filterByKeys

Filter an array by an array of keys.

Usage:

```
<?php

$stack = [
    "ford" => "perfect",
    "marvin" => "android",
    "arthur" => "dent"
];

$keys = [
    "ford",
    "arthur"
];

var_dump(\Comodojo\Foundation\Utils\ArrayOps::filterByKeys($keys, $stack));
```

It returns:

```
array(2) {
    'ford' =>
    string(7) "perfect"
    'arthur' =>
    string(4) "dent"
}
```

6.1.3 ArrayOps::replaceStrict

Perform a selective replace of items only if relative keys are actually defined in source array.

Usage:

```
<?php

$stack = [
    "ford" => "perfect",
    "marvin" => "android",
    "arthur" => "dent"
];

$replace = [
    "marvin" => "robot",
    "tricia" => "mcmillan"
];

var_dump(\Comodojo\Foundation\Utils\ArrayOps::replaceStrict($stack, $replace));
```

It returns:

```
array(3) {
  'ford' =>
  string(7) "perfect"
  'marvin' =>
  string(5) "robot"
  'arthur' =>
  string(4) "dent"
}
```

6.2 Uid generator

Class `\Comodojo\Foundation\Utils\UniqueId` provides 2 different methods to generate an UID (string).

- `UniqueId::generate` generate a random uid, variable length (default 32)
- `UniqueId::generateCustom` generate a random uid that includes provided prefix, , variable length (default 32)

Usage example:

```
<?php

var_dump (\Comodojo\Foundation\Utils\UniqueId::generate(40));

var_dump (\Comodojo\Foundation\Utils\UniqueId::generateCustom('ford', 32));
```

It returns:

```
string(40) "0c7687119b3772a69691b838303f33bdb2c00bcd"

string(32) "ford-47ee5e94f6550d811ab1d007f6f"
```


CHAPTER 7

Data filtering and validation

7.1 Data filtering

Class \Comodojo\Foundation\Validation\DataFilter provides some useful methods to filter data extending (or shortcutting) php funcs.

Included methods are:

- filterInteger: conditional int filter from (\$min, \$max, \$default)
- filterPort: TCP/UDP port filtering
- filterBoolean: boolean filter

Usage example:

```
<?php

$https = 443;
$invalid_port = 10000000;
$default = 8080;

var_dump(\Comodojo\Foundation\Validation\DataFilter::filterPort($https, $default));

var_dump(\Comodojo\Foundation\Validation\DataFilter::filterPort($invalid_port,
    ↴$default));
```

It returns:

```
int(443)

int(8080)
```

7.2 Data validation

Class \Comodojo\Foundation\Validation\DataValidation provides methods to validate data types, optionally applying a custom filter on value itself.

Validation can be invoked via `validate` methods, that accepts input data, data type and filter, or using specific validation methods:

- `validateString`
- `validateBoolean`
- `validateInteger`
- `validateNumeric`
- `validateFloat`
- `validateJson`
- `validateSerialized`
- `validateArray`
- `validateStruct`
- `validateDatetimeIso8601`
- `validateBase64`
- `validateNull`
- `validateTimestamp`

Usage example:

```
<?php

$http = 80;
$https = 443;

$filter = function(int $data) {
    // check if port 80
    return $data === 80;
};

var_dump(\Comodojo\Foundation\Validation\DataValidation::validateInteger($http,
    ↪$filter));

var_dump(\Comodojo\Foundation\Validation\DataValidation::validateInteger($https,
    ↪$filter));
```

It returns:

```
bool(true)

bool(false)
```