arokettu-json Documentation

Release 1.2.0

May 13, 2021

CONTENTS

A wrapper for the standard ext-json with sane defaults.

CHAPTER

ONE

INSTALLATION

```
composer require 'arokettu/json'
```

CHAPTER

TWO

ENCODING

```php
<?php

function \Arokettu\Json\Json::encode(
    mixed $value,
    int|\Arokettu\Json\EncodeOptions $options = JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_
↪UNICODE,
    int $depth = 512,
): string;
```

Main features:

- JSON_THROW_ON_ERROR is enforced
- Two convenience constants:
  - \Arokettu\Json\Json::ENCODE_DEFAULT = JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE
  - \Arokettu\Json\Json::ENCODE_PRETTY = JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT

CHAPTER

THREE

DECODING

```php
<?php

function \Arokettu\Json\Json::decode(
    string $json,
    int|\Arokettu\Json\DecodeOptions $options = 0,
    int $depth = 512,
): mixed;
```

Main features:

- `JSON_THROW_ON_ERROR` is enforced
- Pass `JSON_OBJECT_AS_ARRAY` to get associative arrays
- JSON objects are decoded to instances of `ArrayObject` instead of `stdClass` when parsed as objects

```php
<?php

function \Arokettu\Json\Json::decodeToArray(
    string $json,
    int|\Arokettu\Json\DecodeOptions $options = 0,
    int $depth = 512,
): mixed;
```

Force decoding objects as associative arrays

```php
<?php

function \Arokettu\Json\Json::decodeToObject(
    string $json,
    int|\Arokettu\Json\DecodeOptions int $options = 0,
    int $depth = 512,
): mixed;
```

Force decoding objects as instances of `ArrayObject`

CHAPTER

FOUR

OPTIONS OBJECTS

The library provides 2 classes to manipulate option sets in OOP way:

- `Arokettu\Json\DecodeOptions` for decoding
- `Arokettu\Json\EncodeOptions` for encoding

Objects of both classes are immutable. Any change creates a new instance.

4.1 Constructors

Default constructor:

The default constructor is the least helpful constructor, it can be initialized with json options constants

```php
<?php

$options = new \Arokettu\Json\EncodeOptions(
    JSON_THROW_ON_ERROR | JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE
);
```

Preset constructors:

```php
<?php

// JSON_THROW_ON_ERROR
\Arokettu\Json\DecodeOptions::default();
// JSON_THROW_ON_ERROR | JSON_OBJECT_AS_ARRAY
\Arokettu\Json\DecodeOptions::asArray();

// JSON_THROW_ON_ERROR | JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE
\Arokettu\Json\EncodeOptions::default();
// JSON_THROW_ON_ERROR | JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE | JSON_PRETTY_
↪PRINT
\Arokettu\Json\EncodeOptions::pretty();
```

Builder constructor:

```php
<?php

public static function \Arokettu\Json\DecodeOptions::build(
    int $options = 0,
```

(continues on next page)

(continued from previous page)

```
    ?bool $bigintAsString = null,
    ?bool $objectAsArray = null,
    ?bool $invalidUtf8Ignore = null,
    ?bool $invalidUtf8Substitute = null,
    ?bool $throwOnError = null,
    ?bool $bigint_as_string = null,
    ?bool $object_as_array = null,
    ?bool $invalid_utf8_ignore = null,
    ?bool $invalid_utf8_substitute = null,
    ?bool $throw_on_error = null,
): \Arokettu\Json\DecodeOptions;

public static function \Arokettu\Json\EncodeOptions::build(
    int $options = 0,
    ?bool $hexTag = null,
    ?bool $hexAmp = null,
    ?bool $hexApos = null,
    ?bool $hexQuot = null,
    ?bool $forceObject = null,
    ?bool $numericCheck = null,
    ?bool $prettyPrint = null,
    ?bool $unescapedSlashes = null,
    ?bool $unescapedUnicode = null,
    ?bool $partialOutputOnError = null,
    ?bool $preserveZeroFraction = null,
    ?bool $unescapedLineTerminators = null,
    ?bool $invalidUtf8Ignore = null,
    ?bool $invalidUtf8Substitute = null,
    ?bool $throwOnError = null,
    ?bool $hex_tag = null,
    ?bool $hex_amp = null,
    ?bool $hex_apos = null,
    ?bool $hex_quot = null,
    ?bool $force_object = null,
    ?bool $numeric_check = null,
    ?bool $pretty_print = null,
    ?bool $unescaped_slashes = null,
    ?bool $unescaped_unicode = null,
    ?bool $partial_output_on_error = null,
    ?bool $preserve_zero_fraction = null,
    ?bool $unescaped_line_terminators = null,
    ?bool $invalid_utf8_ignore = null,
    ?bool $invalid_utf8_substitute = null,
    ?bool $throw_on_error = null,
): \Arokettu\Json\EncodeOptions;
```

The builder constructor is made with named parameters in mind. Params exist in both snake case and camel case forms for your preference.

```
<?php

// PHP 8 example
```

(continues on next page)

(continued from previous page)

```php
$options = \Arokettu\Json\EncodeOptions::build(
    throwOnError: true,
    unescapedSlashes: true,
    unescapedUnicode: true,
);

// PHP DI example
$options = (new \DI\Container())->call([\Arokettu\Json\EncodeOptions::class, 'build'], [
    'throw_on_error' => true,
    'unescaped_slashes' => true,
    'unescaped_unicode' => true,
]);

// Initialize options with existing options set to modify it
$options = \Arokettu\Json\EncodeOptions::build(
    JSON_THROW_ON_ERROR | JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE,
    throwOnError: false,
);
```

4.2 Managing options in OOP way

`with*` methods to set their respective flags, `without*` methods to unset them. Objects are immuuable so the methods create new instances of the options.

Full list:

```php
<?php

// Decode setters
function \Arokettu\Json\DecodeOptions::withBigintAsString(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withObjectAsArray(): \Arokettu\Json\DecodeOptions;
function \Arokettu\Json\DecodeOptions::withInvalidUtf8Ignore(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withInvalidUtf8Substitute(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withThrowOnError(): \Arokettu\Json\DecodeOptions;

// Decode unsetters
function \Arokettu\Json\DecodeOptions::withoutBigintAsString(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withoutObjectAsArray(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withoutInvalidUtf8Ignore(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withoutInvalidUtf8Substitute(): \Arokettu\Json\
↪DecodeOptions;
function \Arokettu\Json\DecodeOptions::withoutThrowOnError(): \Arokettu\Json\
↪DecodeOptions;
```

(continues on next page)

(continued from previous page)

```
// Encode setters
function \Arokettu\Json\EncodeOptions::withHexTag(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withHexAmp(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withHexApos(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withHexQuot(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withForceObject(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withNumericCheck(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withPrettyPrint(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withUnescapedSlashes(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withUnescapedUnicode(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withPartialOutputOnError(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withPreserveZeroFraction(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withUnescapedLineTerminators(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withInvalidUtf8Ignore(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withInvalidUtf8Substitute(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withThrowOnError(): \Arokettu\Json\EncodeOptions;

// Encode unsetters
function \Arokettu\Json\EncodeOptions::withoutHexTag(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutHexAmp(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutHexApos(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutHexQuot(): \Arokettu\Json\EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutForceObject(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutNumericCheck(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutPrettyPrint(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutUnescapedSlashes(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutUnescapedUnicode(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutPartialOutputOnError(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutPreserveZeroFraction(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutUnescapedLineTerminators(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutInvalidUtf8Ignore(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutInvalidUtf8Substitute(): \Arokettu\Json\
↪EncodeOptions;
function \Arokettu\Json\EncodeOptions::withoutThrowOnError(): \Arokettu\Json\
↪EncodeOptions;
```

Example:

```php
<?php

$options = \Arokettu\Json\EncodeOptions::default()
    ->withPrettyPrint()
    ->withoutThrowOnError()
;
```

4.3 Value getters

```php
<?php

$options->value(); // get integer value
$options->toInt(); // alias of value()
$options->toString(); // export options list as a conjunction of base ext-json constants to a string
```

Int getter can be used with vanilla `ext-json` methods:

```php
<?php

echo json_encode($value, \Arokettu\Json\EncodeOptions::pretty()->value());
```

String getter can be useful for debug or code generation

```php
<?php

$pretty = \Arokettu\Json\EncodeOptions::pretty()->toString();
// returns "JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE | JSON_THROW_ON_ERROR"

$php = <<<PHP
    <?php
    return json_encode(\$value, {$pretty});
    PHP;
// generates:
//  <?php
//  return json_encode($value, JSON_PRETTY_PRINT | JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE | JSON_THROW_ON_ERROR);
```

CHAPTER

FIVE

LICENSE

The library is available as open source under the terms of the MIT License.