KepplerPl_Scheme Documentation

Release v2.0.0

Andrei Popa

Jan 04, 2019

Contents:

CHAPTER 1

Overview

1.1 Requirements

1. PHP >= 7.1.x

1.2 Installation

The recommended way to install this package is with Composer. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project needs and installs them into your project.

```
# Install Composer
curl -sS https://getcomposer.org/installer | php
```

```
composer require keppler/url
```

After installing, you need to require Composer's autoloader:

```
require 'vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

1.3 License

Licensed using the MIT license.

1.4 Reporting an issue

Please report all issues here

CHAPTER 2

Quickstart

2.1 Parser and Builder

This package is split into 2 independent pieces. These pieces are also split into several other pieces.

Note: The Scheme.php will be referred to as Parser in the documentation.

2.1.1 Parser

The Parser is the entry point for parsing a url. It's immutable, meaning you cannot change it once it is created.

The scheme is split into schemes such as ftp, http, https, mailto, etc. Each scheme is used to parse a single url type, as you might have guessed.

```
require 'vendor/autoload.php';

$url =  'ftp://user:password@host:123/path';

$scheme = Scheme::ftp($url);

print_r($scheme->all());

...

Array
(
    [scheme] => ftp
    [user] => user
    [password] => password
    [host] => host
    [port] => 123
```

(continues on next page)

(continued from previous page)

```
    [path] => Array
        (
            [0] => path
        )

)
```

At the time of this writing the parser supports 4 schemes: FTP, HTTPS, HTTP, and MAILTO

2.1.2 Builder

The Builder.php class is the entry point for modifying a url or simply creating one from scratch. If you choose to build from an existing url you must pass it a Parser instance with the appropriate scheme.

At the time of this writing the Builder supports 4 schemes: FTP, HTTPS, HTTP, and MAILTO

```
require 'vendor/autoload.php';

$url =  'ftp://user:password@host:123/path';

$ftpScheme = Scheme::ftp($url);
$builder = Builder::ftp($ftpScheme);

$builder->setHost('example.com')
    ->setPassword('hunter2')
    ->setPort(5);

print_r($builder->raw());
...
ftp://user:hunter2@example.com:5/path/

print_r($builder->encoded());
...
ftp://user:hunter2@example.com:5/path/to+encode/ // notice the extra +
```

Note: Both the Parser and the Builder can be used independently.

Each supported scheme can also be used independently without the Builder or the Parser. Examples bellow.

2.1.3 Independent usage

Assuming you don't want to use the Parser/Builder classes directly you can choose not to.

Each scheme supported can be used independently of the Parser/Builder.

```
$ftpUrl = 'ftp://user:password@host:123/path';

$ftpImmutable = new FtpImmutable($ftpUrl);

echo $ftpImmutable->raw();
```

```
$ftpBuilder = new FtpBuilder();

$ftpBuilder->setHost('host')
    ->setPassword('hunter2')
    ->setPort(987)
    ->setUser('hunter');

$ftpBuilder->getPathBag()
    ->set(0, 'path')
    ->set(1, 'new path');

echo $ftpBuilder->raw(); // ftp://hunter:hunter2@host:987/path/new path/

echo $ftpBuilder->encoded(); // ftp://hunter:hunter2@host:987/path/new+path/
```

CHAPTER 3

Parser

3.1 Usage and Interface

3.1.1 Usage

Warning: PLEASE READ!

The parser makes absolutely no promises regarding the validity of the scheme nor does it try to parse severely malformed urls.

Passing such urls to the parser will most likely result in an error.

If a query or path is given to a scheme that doesn't support it, it will be discarded

Some url schemes MAY not have information in the path/query bag since some urls can simply not have a path or a query. For example the mailto scheme may not have a query or a path, or both. The ftp scheme simply doesn't support a query so the parse will automatically discard it if one is given.

The path and/or query bags will ALWAYS exist but they may not contain any information.

Note: The classes found in the Parser can be used independently too. See the Quickstart link bellow.

Quickstart

The Scheme.php class is used as the parser. Any Parser instance is immutable, meaning you cannot change it once it has been created.

The usage is straight forward:

```
$url =  'ftp://user:password@host:123/path';
$ftp = Scheme::ftp($url);
```

```
$url = 'https://john.doe@www.example.com:123/forum/questions/?tag=networking&
↪order=newest#top';
$https = Scheme::https($url);
```

```
$url = 'http://john.doe@www.example.com:123/forum/questions/?tag=networking&
↪order=newest#top';
$http = Scheme::http($url);
```

```
$url = 'mailto:path@email.com,path2@email.com?to=email@example.com,email2@example.com&
↪cc=email3@example.com,email4@example.com&bcc=email4@example.com,email5@example.com&
↪subject=Hello&body=World';
$mailto = Scheme::mailto($url);
```

The parser also contains bags for the query or path. Given that it can exist within the given scheme.

Some schemes don't have a path or a query. For example the FTP scheme officially does not support a query, thus the Parser doesn't support one either.

```
$url =  'ftp://user:password@host:123/path';
$ftp = Scheme::ftp($url);

// Only path
$ftp->getPathBag()->...;
```

Other types of urls will support both a path and a query bag.

```
$url = 'https://john.doe@www.example.com:123/forum/questions/?tag=networking&
↪order=newest#top';
$https = Scheme::https($url);

$https->getPathBag()->...;
$https->getQueryBag()->...;
```

3.1.2 Parser Interface

All parsers implement the ImmutableSchemeInterface which has the following functions

```
// Returns all the components of the scheme including any bags in the form of an array
// Will always return an array, even if empty.

public function all(): array;

// Returns raw unaltered url

public function raw(): string

// Returns the scheme associated with the class instance

public function getScheme(): string;
```

3.1.3 Bags Interface

All immutable bags(query and path) implement the ImmutableBagInterface which has the following functions

```
// Returns all the components of the query or path

public function all(): array;
```

```
// Return the raw unaltered query or path

public function raw(): string;
```

3.2 Mailto

The mailto scheme has a path and a query bag along side the default interface options

The mailto scheme class does it's best to keep in accordance with https://tools.ietf.org/html/rfc6068

The mailto immutable has no other functions except the default implementations and getters for the bags.

3.2.1 The query bag

The mailto scheme can have a query consisting of: to recipients, cc recipients, bcc recipients, body, and subject.

```
$url = 'mailto:path@email.com,path2@email.com?to=email@example.com,email2@example.com&
↪cc=email3@example.com,email4@example.com&bcc=email4@example.com,email5@example.com&
↪subject=Hello&body=World';
$mailto = Scheme::mailto($url);
echo $mailto->getQueryBag()->firstInTo(); // email@example.com
echo $mailto->getQueryBag()->lastInTo(); // email2@example.com
echo $mailto->getQueryBag()->hasInTo('email@example.com'); // true
echo $mailto->getQueryBag()->hasInTo('not_in_to@example.com'); // false
```

The same goes for CC and BCC functions with the only difference being the suffix of the function

Besides the to, cc, and bcc functions getters are available for subject and body

```
public function getSubject(): string

public function getBody(): string

public function getBcc(): array

public function getCc(): array

public function getTo(): array
```

3.2.2 The path bag

Much like the query bag, the path bag comes with its own functions

```
public function first()

public function last()

public function hasInPath(string $value): bool
```

(continues on next page)

(continued from previous page)

```
public function getPath(): array
```

Due to the simplicity of the path in mailto schemes the path bag is not very feature rich.

3.3 Http and Https

The http and https schemes have a path and a query bag along side the default interface options

The http and https scheme classes do their best to keep in accordance with https://tools.ietf.org/html/rfc3986

Note: Due to major similarities between the 2 schemes there is a single section dedicated to both.

HOWEVER each scheme has its own dedicated parser.

Besides the default interface implementation the http and https immutable classes have the following functions

```
public function getAuthority(): string

public function getUser(): string

public function getPassword(): string

public function getHost(): string

public function getPort(): ?int

public function getFragment(): string

public function getQueryBag(): HttpImmutableQuery

public function getPathBag(): HttpImmutablePath
```

3.3.1 The query bag

Besides the default interface implementation the http/https immutable bags classes have the following functions

```
$url = 'http://john:password@www.example.com:123/forum/questions 10/?&
↪tag[]=networking&tag[]=cisco&order=newest#top';

$scheme = Scheme::http($url);

var_dump($scheme->getQueryBag()->get('tag'));

...

Array
(
    [0] => networking
    [1] => cisco
)
```

```
public function get($key)

public function has($key): bool

public function first(): ?array

public function last(): ?string
```

3.3.2 The path bag

Besides the default interface implementation the http/https immutable bags classes have the following functions

```
$url = 'http://john:password@www.example.com:123/forum/questions 10/?&
↪tag[]=networking&tag[]=cisco&order=newest#top';

$scheme = Scheme::http($url);

var_dump($scheme->getPathBag()->get(0));
...
string(5) "forum"

$scheme->getPathBag()->get(10);
...
Fatal error:  Uncaught Keppler\Url\Exceptions\ComponentNotFoundException: Component␣
↪with index "10" does not exist in␣
↪Keppler\Url\Scheme\Schemes\Http\Bags\HttpImmutablePath
```

```
public function first(): ?string

public function last(): ?string

public function get(int $key)

public function has(int $key): bool
```

3.4 Ftp

The ftp parser has only a path bag along side the default interface options

The ftp class does its best to keep in accordance with https://tools.ietf.org/html/rfc3986

Besides the default interface implementation the ftp immutable class has the following functions

```
public function getPathBag(): FtpImmutablePath

public function getUser(): string

public function getPassword(): string

public function getHost(): string

public function getPort(): ?int
```

3.4.1 The path bag

Besides the default interface implementation the ftp immutable bag class has the following functions

```
public function first(): ?string

public function last(): ?string

public function get(int $key)

public function has(int $key): bool
```

CHAPTER 4

Builder

4.1 Usage and Interface

4.1.1 Usage

Note: The classes found in the Builder can be used independently too. See the Quickstart link bellow.

Quickstart

The Builder class is mutable. It will accept an Immutable class of the corresponding scheme.

The usage is straight forward:

```
$ftpUrl = 'ftp://user:password@host:123/path';
$ftpImmutable = Scheme::ftp($ftpUrl);

$builder = Builder::ftp($ftpImmutable);

$builder->setUser('JohnDoe')
    ->setHost('example.com')
    ->setPassword('hunter2')
    ->setPort(987);

$builder->getPathBag()
    ->set(0, 'new path value')
    ->set(1, 'another path value');

echo $builder->raw(); // ftp://JohnDoe:hunter2@example.com:987/new path value/another␣
↪path value/

echo $builder->encoded(); // ftp://JohnDoe:hunter2@example.com:987/new+path+value/
↪another+path+value/
```

```
$url = 'https://john.doe@www.example.com:123/forum/questions/?tag=networking&
↪order=newest#top';
$httpsImmutable = Scheme::https($url);

$builder = Builder::https($httpsImmutable);
...
```

```
$url = 'http://john.doe@www.example.com:123/forum/questions/?tag=networking&
↪order=newest#top';
$httpImmutable = Scheme::http($url);

$builder = Builder::https($httpsImmutable);
...
```

```
$url = 'mailto:path@email.com,path2@email.com?to=email@example.com,email2@example.com&
↪cc=email3@example.com,email4@example.com&bcc=email4@example.com,email5@example.com&
↪subject=Hello&body=World';
$mailtoImmutable = Scheme::mailto($url);

$builder = Builder::https($mailtoImmutable);
...
```

The builder also contains bags for the query or path. Given that it can exist within the given scheme.

Some schemes don't have a path or a query. For example the FTP scheme officially does not support a query, thus the Builder doesn't support one either.

```
$ftpUrl = 'ftp://user:password@host:123/path';
$ftpImmutable = Scheme::ftp($ftpUrl);

$builder = Builder::ftp($ftpImmutable);

// Only path
$builder->getPathBag()->...;
```

Other types of urls will support both a path and a query bag.

```
$url = 'https://john.doe@www.example.com:123/forum/questions/?tag=networking&
↪order=newest#top';
$httpsImmutable = Scheme::https($url);

$builder = Builder::https($httpsImmutable);

$builder->getPathBag()->...;
$builder->getQueryBag()->...;
```

4.1.2 Builder Interface

All builders implement the ImmutableSchemeInterface which has the following functions

```
// Returns all the components of the scheme including any bags in the form of an array
// Will always return an array, even if empty.

public function all(): array;

```

(continues on next page)

(continued from previous page)

```
// Returns raw unaltered url

public function raw(): string

// Returns the scheme associated with the class instance

public function getScheme(): string;

// Builds the url either encoded or not
public function build(bool $urlEncode = false): string;
```

Note: The build function is an alias for raw() and encoded() with the $urlEncode specified as either true or false

4.1.3 Bags Interface

All mutable bags(query and path) implement the MutableBagsInterface which has the following functions

```
// Returns all the components of the query or path

public function all(): array;
```

```
// Returns the encoded query or path string

public function encoded(): string;

// Return the raw unaltered query or path

public function raw(): string;

// Checks weather a given bag or path has a certain key

public function has($key): bool;

// Returns a given key

public function get($key);

// Sets a new entry in the path or query

public function set($key, $value);

// Returns all the components of the query or path

public function all(): array;
```

4.2 Mailto

The mailto builder has a path and a query bag along side the default interface options

The mailto builder class does it's best to keep in accordance with https://tools.ietf.org/html/rfc6068

The mailto immutable has no other functions except the default implementations and getters for the bags.

4.2.1 The query bag

The mailto scheme can have a query consisting of: to recipients, cc recipients, bcc recipients, body, and subject.

Besides the getter functions specified in the previous chapter the builder has the following functions available.

```
public function putInTo(string $value): self

public function putInCc(string $value): self

public function putInBcc(string $value): self

public function forgetFromTo($keyOrValue): self

public function forgetFromCc($keyOrValue): self

public function forgetFromBcc($keyOrValue): self

public function forgetTo(): self

public function forgetCc(): self

public function forgetBcc(): self

public function forgetSubject(): self

public function forgetBody(): self

public function toHas(string $value): bool

public function ccHas(string $value): bool

public function bccHas(string $value): bool

public function forgetAll(): self

// Returns only the specified class properties(in this case)
public function only(string ...$args): array
```

Note: Functions such as get and set in this particular case will search for a class property rather than a query component

```
$url
    = 'mailto:path@email.com,path2@email.com?to=email@example.com,email2@example.com'.
    '&cc=email3@example.com,email4@example.com'.
    '&bcc=email4@example.com,email5@example.com'.
    '&subject=Hello'.
    '&body=World';

$mailto = Scheme::mailto($url);
$builder = Builder::mailto($mailto);

var_dump($builder->getQueryBag()->only('cc', 'bcc'));
...
Array
(
```

(continues on next page)

(continued from previous page)

```
    [cc] => Array
        (
            [0] => email3@example.com
            [1] => email4@example.com
        )

    [bcc] => Array
        (
            [0] => email4@example.com
            [1] => email5@example.com
        )

)

$builder->getQueryBag()->forgetFromBcc('email5@example.com');
$builder->getQueryBag()->forgetFromBcc(0);

var_dump($builder->getQueryBag()->only('cc', 'bcc'));
...
Array
(
    [cc] => Array
        (
            [0] => email3@example.com
            [1] => email4@example.com
        )

    [bcc] => Array
        (
        )

)
```

4.2.2 The path bag

Much like the query bag, the path bag comes with its own functions

```
public function setPath(array $path): self

public function getPath(): array

public function append(string $value): self

public function prepend(string $value): self

public function putInBetween(string $value, string $first = null, string $last =␣
↪null): self

public function putBefore(string $before, string $value) : self

public function first()

public function last()

public function putAfter(string $after, string $value): self
```

(continues on next page)

(continued from previous page)

```
public function forget(string ...$args): self

public function forgetAll(): self

public function only(string ...$args): array
```

```
$url
    = 'mailto:path@email.com,path2@email.com?to=email@example.com,email2@example.com'.
    '&cc=email3@example.com,email4@example.com'.
    '&bcc=email4@example.com,email5@example.com'.
    '&subject=Hello'.
    '&body=World';

$mailto = Scheme::mailto($url);
$builder = Builder::mailto($mailto);

$builder->getPathBag()->putInBetween('new_value@test.com', 'path@email.com');

var_dump($builder->getPathBag()->all());
...
Array
(
    [0] => path@email.com
    [1] => new_value@test.com
    [2] => path2@email.com
)

$builder->getPathBag()->putAfter('new_value@test.com', 'after_new_value@test.com');

var_dump($builder->getPathBag()->all());
...
Array
(
    [0] => path@email.com
    [1] => new_value@test.com
    [2] => after_new_value@test.com
    [3] => path2@email.com
)
```

4.3 Http and Https

The http and https schemes have a path and a query bag along side the default interface options

The http and https scheme classes do their best to keep in accordance with https://tools.ietf.org/html/rfc3986

Note: Due to major similarities between the 2 schemes there is a single section dedicated to both.

HOWEVER each scheme has its own dedicated builder.

```
public function getAuthority(): string

public function getUser(): string
```

(continues on next page)

(continued from previous page)

```
public function getPassword(): string

public function getHost(): string

public function getPort(): ?int

public function getFragment(): string

public function getQueryBag(): HttpImmutableQuery

public function getPathBag(): HttpImmutablePath

public function setUser(string $user): self

public function setPassword(string $password): self

public function setHost(string $host): self

public function setPort(int $port): self

public function setFragment(string $fragment): self
```

4.3.1 The query bag

Besides the default interface implementation the http/https mutable bags classes have the following functions

```
public function first(): ?array

public function last()

public function forget(string ...$args): self

public function forgetAll(): self

public function only(string ...$args): array
```

4.3.2 The path bag

Besides the default interface implementation the http/https bags bags classes have the following functions

```
public function getPath(): array

public function first(): ?string

public function last(): ?string

public function append(string $value): self

public function prepend(string $value): self

public function putInBetween(string $value, string $first = null, string $last =␣
↪null): self
```

(continues on next page)

(continued from previous page)

```
public function putBefore(string $before, string $value) : self

public function putAfter(string $after, string $value): self

public function forget(string ...$args): self

public function forgetAll(): self

public function only(string ...$args): array
```

4.4 Ftp

The ftp builder has only a path bag along side the default interface options

The ftp class does its best to keep in accordance with https://tools.ietf.org/html/rfc3986

Besides the default interface implementation the ftp mutable class has the following functions

```
public function getPathBag(): FtpMutablePath

public function getUser(): string

public function getPassword(): string

public function getHost(): string

public function getPort(): int

public function setUser(string $user): self

public function setPassword(string $password): self

public function setHost(string $host): self

public function setPort(int $port): self
```

4.4.1 The path bag

Besides the default interface implementation the ftp immutable bag class has the following functions

```
public function getPath(): array

public function first(): ?string

public function last()

public function append(string $value): self

public function prepend(string $value): self

public function putInBetween(string $value, string $first = null, string $last =␣
↪null): self
```

(continues on next page)

(continued from previous page)

```
public function putBefore(string $before, string $value) : self

public function putAfter(string $after, string $value): self

public function forget(string ...$args): self

public function forgetAll(): self

public function only(string ...$args): array
```