### PSR-7 Usage

All PSR-7 applications comply with these interfaces They were created to establish a standard between middleware implementations.

RequestInterface, ServerRequestInterface, ResponseInterface extend MessageInterface because the Request and the Response are HTTP Messages. When using ServerRequestInterface, both RequestInterface and Psr\Http\Message\MessageInterface methods are considered.

The following examples will illustrate how basic operations are done in PSR-7.

##### Examples

For this examples to work (at least) a PSR-7 implementation package is required. (eg: zendframework/zend-diactoros, guzzlehttp/psr7, slim/slim, etc) All PSR-7 implementations should have the same behaviour.

The following will be assumed: $request is an object of Psr\Http\Message\RequestInterface and

$response is an object implementing Psr\Http\Message\RequestInterface

### Working with HTTP Headers

#### Adding headers to response:

$response->withHeader('My-Custom-Header', 'My Custom Message');

#### Appending values to headers

$response->withAddedHeader('My-Custom-Header', 'The second message');

#### Checking if header exists:

$request->hasHeader('My-Custom-Header'); // will return false

$response->hasHeader('My-Custom-Header'); // will return true

Note: My-Custom-Header was only added in the Response

#### Getting comma-separated values from a header (also applies to request)

// getting value from request headers

$request->getHeaderLine('Content-Type'); // will return: "text/html; charset=UTF-8"

// getting value from response headers

$response->getHeaderLine('My-Custom-Header'); // will return: "My Custom Message; The second message"

#### Getting array of value from a header (also applies to request)

// getting value from request headers

$request->getHeader('Content-Type'); // will return: ["text/html", "charset=UTF-8"]

// getting value from response headers

$response->getHeader('My-Custom-Header'); // will return: ["My Custom Message", "The second message"]

#### Removing headers from HTTP Messages

// removing a header from Request, removing deprecated "Content-MD5" header

$request->withoutHeader('Content-MD5');

// removing a header from Response

// effect: the browser won't know the size of the stream

// the browser will download the stream till it ends

$response->withoutHeader('Content-Length');

### Working with HTTP Message Body

When working with the PSR-7 there are two methods of implementation:

#### 1. Getting the body separately

This method makes the body handling easier to understand and is useful when repeatedly calling body methods. (You only call getBody() once). Using this method mistakes like $response->write() are also prevented.

$body = $response->getBody();

// operations on body, eg. read, write, seek

// ...

// replacing the old body

$response->withBody($body);

// this last statement is optional as we working with objects

// in this case the "new" body is same with the "old" one

// the $body variable has the same value as the one in $request, only the reference is passed

#### 2. Working directly on response

This method is useful when only performing few operations as the $request->getBody() statement fragment is required

$response->getBody()->write('hello');

### Getting the body contents

The following snippet gets the contents of a stream contents.

Note: Streams must be rewinded, if content was written into streams, it will be ignored when calling getContents() because the stream pointer is set to the last character, which is \0 - meaning end of stream.

$body = $response->getBody();

$body->rewind(); // or $body->seek(0);

$bodyText = $body->getContents();

Note: If $body->seek(1) is called before $body->getContents(), the first character will be ommited as the starting pointer is set to 1, not 0. This is why using $body->rewind() is recommended.

### Append to body

$response->getBody()->write('Hello'); // writing directly

$body = $request->getBody(); // which is a `StreamInterface`

$body->write('xxxxx');

### Prepend to body

Prepending is different when it comes to streams. The content must be copied before writing the content to be prepended. The following example will explain the behaviour of streams.

// assuming our response is initially empty

$body = $repsonse->getBody();

// writing the string "abcd"

$body->write('abcd');

// seeking to start of stream

$body->seek(0);

// writing 'ef'

$body->write('ef'); // at this point the stream contains "efcd"

#### Prepending by rewriting separately

// assuming our response body stream only contains: "abcd"

$body = $response->getBody();

$body->rewind();

$contents = $body->getContents(); // abcd

// seeking the stream to beginning

$body->rewind();

$body->write('ef'); // stream contains "efcd"

$body->write($contents); // stream contains "efabcd"

Note: getContents() seeks the stream while reading it, therefore if the second rewind() method call was not present the stream would have resulted in abcdefabcd because the write() method appends to stream if not preceeded by rewind() or seek(0).

#### Prepending by using contents as a string

$body = $response->getBody();

$body->rewind();

$contents = $body->getContents(); // efabcd

$contents = 'ef'.$contents;

$body->rewind();

$body->write($contents);