

Web Scraping with PHP

Second Edition

by Matthew Turland



Web Scraping with PHP

2nd Edition

by Matthew Turland



php[**architect**] edition

Web Scraping with PHP

Contents Copyright ©2019 Matthew Turland—All Rights Reserved

Book and cover layout, design and text Copyright ©2019 musketeers.me, LLC. and its predecessors—All Rights Reserved. Print and Digital copies available from <https://www.phparch.com/books/>.

php[architect] edition published: August 2019

Print ISBN:	978-1-940111-67-4
PDF ISBN:	978-1-940111-68-1
ePub ISBN:	978-1-940111-60-8
Mobi ISBN	978-1-940111-70-4

Produced & Printed in the United States

No part of this book may be reproduced, stored in a public retrieval system, or publicly transmitted in any form or by means without the prior written permission of the publisher, except in the case of brief quotations embedded in critical reviews or articles.

Disclaimer

Although every effort has been made in the preparation of this book to ensure the accuracy of the information contained therein, this book is provided "as-is" and the publisher, the author(s), their distributors and retailers, as well as all affiliated, related or subsidiary parties take no responsibility for any inaccuracy and any and all damages caused, either directly or indirectly, by the use of such information. We have endeavored to properly provide trademark information on all companies and products mentioned in the book by the appropriate use of capitals. However, we cannot guarantee the accuracy of such information.

musketeers.me, the musketeers.me logo, php[architect], the php[architect] logo are trademarks or registered trademarks of musketeers.me, LLC, its assigns, partners, predecessors and successors.

All other trademarks are the property of the respective owners.

Written by
Matthew Turland

Managing Editor
Oscar Merida

Editor
Kara Ferguson

Published by
musketeers.me, LLC.
4627 University Dr
Fairfax, VA 22030 USA

240-348-5PHP (240-348-5747)
info@phparch.com
www.phparch.com

Table of Contents

1. Introduction	1
Intended Audience	1
How to Read This Book	2
Web Scraping Defined	2
Applications of Web Scraping	3
Appropriate Use of Web Scraping	3
Legality of Web Scraping	3
Topics Covered	4
2. HTTP	5
Requests	6
Responses	11
Headers	12
Evolution of HTTP	19

TABLE OF CONTENTS

3. HTTP Streams Wrapper	21
Simple Request and Response Handling	22
Stream Contexts and POST Requests	23
Error Handling	24
HTTP Authentication	26
More Options	26
4. cURL Extension	27
Simple Request and Response Handling	28
Contrasting GET and POST	29
Setting Options	30
Analyzing Results	31
Handling Headers	33
Debugging	33
Cookies	34
HTTP Authentication	35
Security	35
Redirection	36
Referrers	37
Content Caching	37
User Agents	38
Byte Ranges	38
DNS Caching	39
Timeouts	40
Basic Request Pooling	40
More Efficient Request Pooling	42
Libraries	44

5. pecl_http Extension	45
Installation	46
GET Requests	46
POST Requests	47
Request Options	48
Handling Headers	49
Debugging	50
Timeouts	50
Content Encoding	50
Cookies	51
HTTP Authentication	52
Redirection and Referrers	52
Content Caching	53
User Agents	53
Byte Ranges	53
Request Pooling	54
6. Guzzle	55
Simple Request and Response Handling	56
POST Requests	56
Handling Headers	57
Analyzing Responses	58
Request Objects	59
Connectivity	60
Debugging	60
Cookies	61
Redirection	62
Authentication	63
Security	63

TABLE OF CONTENTS

Asynchronous Requests	64
Concurrent Requests	66
7. Zend Framework	67
Basic Requests	68
Responses	71
URL Handling	73
Custom Headers	73
Configuration	75
Connectivity	75
Debugging	76
Cookies	76
Redirection	80
User Agents	80
HTTP Authentication	80
8. Rolling Your Own	81
Sending Requests	81
Parsing Responses	84
Transfer-Encoding	84
Content Encoding	85
Timing	86
9. Tidy Extension	87
Validation	88
Tidy	89
Input	89
Configuration	89
Options	91

Debugging	92
Output	93
10. DOM Extension	95
Types of Parsers	96
Loading Documents	96
Tree Terminology	97
Elements and Attributes	98
Locating Nodes	98
XPath and DOMXPath	100
Absolute Addressing	100
Relative Addressing	102
Addressing Attributes	102
Unions	102
Conditions	103
Using PHP Functions	103
Resources	105
11. SimpleXML Extension	107
Loading a Document	108
Accessing Elements	108
Accessing Attributes	109
Comparing Nodes	110
DOM Interoperability	110
XPath	111

12. XMLReader Extension	113
Loading a Document	114
Iteration	115
Nodes	115
Elements and Attributes	116
readString() Availability	118
DOM Interoperation	120
Closing Documents	120
13. CSS Selector Libraries	121
Reason to Use Them	122
Basics	122
Hierarchical Selectors	123
Basic Filters	124
Content Filters	125
Attribute Filters	125
Child Filters	126
Form Filters	127
Libraries	127
14. Symfony Libraries	131
CssSelector	131
DomCrawler	133
BrowserKit	136
Goutte	136
HttpClient	137
Panther	138

15. PCRE Extension	141
Pattern Basics	142
Anchors	142
Alternation	143
Repetition and Quantifiers	144
Subpatterns	145
Matching	145
Escaping	147
Escape Sequences	147
Modifiers	149
16. Practical Applications	151
Crawler	151
Scraper	155
Acceptance Tests	159
A. Legality of Web Scraping	165
Index	169

Sample

Chapter

3

HTTP Streams Wrapper

At this point, you should be reasonably well-acquainted with some of the general concepts involved in using an HTTP client. The next chapters review some of the more popular mainstream client libraries, including common use cases and the advantages and disadvantages of each. The client covered in this chapter is the HTTP streams wrapper^[1].

PHP 4.3 saw the addition of the Streams extension to the core. According to the related section of the PHP manual, the intention was to provide “a way of generalizing file, network, data compression, and other operations which share a common set of functions and uses.” One of the concepts that streams introduced was the **wrapper**. The job of a wrapper is to define how a stream handles communications in a specific protocol or using a specific encoding. One such protocol for which a wrapper is available is HTTP.

The primary advantages of the HTTP streams wrapper are its ease of use and availability. Its API is minimal; it’s easy and quick to get something simple working. The HTTP streams wrapper is part

[1] HTTP streams wrapper: <http://php.net/wrappers.http>

3. HTTP STREAMS WRAPPER

of the PHP core; thus, it's available in all PHP installations, as opposed to an optional extension that may not be, and has no other installation requirements.

The disadvantage of the HTTP streams wrapper is its minimal feature set. It gives you the ability to send HTTP requests without having to construct them entirely on your own (by specifying the body and optionally any headers you want to add) and access data in the response. That's about it. The ability to debug requests is one example of a feature it does not include at the time of this writing.

The fact that the wrapper is C code is a bit of a double-edged sword. On the positive side, there is a significant performance difference between C code and PHP code (though it's more noticeable in a high load environment). On the negative side, you have to either know C or depend on the community to deliver patches for any issues which may arise. This also applies to extensions written in C covered in later sections.

Simple Request and Response Handling

Here's a simple example of the HTTP streams wrapper in action.

```
$response = file_get_contents('http://example.com');  
print_r($http_response_header);
```

Some notes:

- You must enable the `allow_url_fopen` PHP configuration setting for this to work, it's enabled in most environments.
- In this example, `file_get_contents()`^[2] makes a GET request for the specified URL `http://example.com`.
- `$response` will contain the response body after the call to the `file_get_contents()` function completes.
- `$http_response_header` is implicitly populated with the HTTP response status line and headers after the `file_get_contents()` call because it uses the HTTP streams wrapper *within the current scope*.

While this example does work, it violates a core principle of good coding practices: no unexpected side effects. The origin of `$http_response_header` is not entirely obvious because PHP populates it implicitly. It's also more restrictive because the variable is unavailable outside the scope containing the call to `file_get_contents()`. Here's a better way to get access to the same data from the response headers.

```
$handle = fopen('http://example.com', 'r');  
$response = stream_get_contents($handle);  
$meta = stream_get_meta_data($handle);  
print_r($meta['wrapper_data']);
```

[2] `file_get_contents()`: http://php.net/file_get_contents

Let's step through this.

1. `fopen()` opens a connection to the URL `http://example.com`; the resource `$handle` references a stream for that connection.
2. `stream_get_contents()` reads the remaining data on the stream pointed to by the `$handle` resource into `$response`.
3. `stream_get_meta_data()` reads metadata for the stream pointed to by the `$handle` resource into `$meta`.
4. At this point, `$meta['wrapper_data']` contains the same array as `$http_response_header` would within the current scope. You can call `stream_get_metadata()` with `$handle` in any scope in which the latter is available. This makes it more flexible than `$http_response_header`.

Stream Contexts and POST Requests

Another concept introduced by streams is the **context**^[3], which is a set of configuration options used in a streams operation. `stream_context_create()` receives an associative array of context options and their corresponding values and returns a context. When using the HTTP streams wrapper, one use of contexts is to make POST requests, as the wrapper uses the GET method by default.

Listing 3.1

```

1. <?php
2. $context = stream_context_create([
3.     'http' => [
4.         'method' => 'POST',
5.         'header' => implode(
6.             "\r\n", [
7.                 'Content-Type: application/x-www-form-urlencoded',
8.                 'Referer: http://example.com'
9.             ]
10.        ),
11.        'content' => http_build_query([
12.            'param1' => 'value1',
13.            'param2' => 'value2'
14.        ]),
15.    ]
16. ]);
17.
18. $response = file_get_contents(
19.     'http://example.com/process', false, $context
20. );

```

[3] context: <http://php.net/context.http>

3. HTTP STREAMS WRAPPER

Here is a walk-through of the example in Listing 3.1.

- 'http' is the streams wrapper used.
- 'POST' is the HTTP method of the request.
- The 'header' stream context setting references a string containing HTTP header key-value pairs, in this case for the Content-Type and Referer HTTP headers. The Content-Type header indicates the request body data is URL-encoded. If you need to set more than one custom header, you must separate them with a carriage return-line feed sequence ("\r\n" also known as a CRLF). `implode()`^[4] is useful for this if you store key-value pairs for headers.
- `http_build_query()`^[5] constructs the body of the request. It can also construct query strings of URLs for GET requests. One useful aspect is that it automatically handles encoding key-value pairs and delimiting them with an ampersand.
- `http://example.com/process` is the URL of the requested resource.
- `file_get_contents()`^[6] executes the request using options from the context `$context` created using `stream_context_create()`^[7].
- `$response` receives the body of the response returned by `file_get_contents()`.

Error Handling

Before PHP 5.3.0, an HTTP streams wrapper operation resulting in an HTTP error response (i.e., a 4xx or 5xx status code) emits a PHP-level warning. This warning contains the HTTP version, the status code, and the status code description. The function calls for such operations generally return `false` as a result, and leave you without a stream resource to check for more information. Listing 3.2 is an example of how to get what data you can.

Listing 3.2

```
1. <?php
2. function error_handler($errno, $errstr, $errfile, $errline, array $errcontext) {
3.     // $errstr will contain something like this:
4.     // fopen(http:_example.com/404): failed to open stream:
5.     // HTTP request failed! HTTP/1.0 404 Not Found
6.     if ($httperr = strstr($errstr, 'HTTP/')) {
7.         // $httperr will contain HTTP/1.0 404 Not Found in the case
8.         // of the above example, do something useful with that here
9.     }
10. }
11.
12. set_error_handler('error_handler', E_WARNING);
13.
```

[4] `implode()`: <http://php.net/implode>

[5] `http_build_query()`: http://php.net/http_build_query

[6] `file_get_contents()`: http://php.net/file_get_contents

[7] `stream_context_create()`: http://php.net/stream_context_create

```

14. // If the following statement fails, $stream will be assigned
15. // false and error_handler will be called automatically
16. $stream = fopen('http://example.com/404', 'r');
17.
18. // If error_handler() does not terminate the script, control
19. // will be returned here once it completes its execution
20. restore_error_handler();

```

This situation improved somewhat in PHP 5.3 with the addition of the `ignore_errors` context setting. When you set this setting to `true`, PHP treats operations resulting in errors the same way as successful operations and emits no warnings. Listing 3.3 is an example of what it might look like.

Listing 3.3

```

1. <?php
2. $context = stream_context_create([
3.     'http' => [
4.         'ignore_errors' => true
5.     ]
6. ]);
7.
8. $stream = fopen('http://example.com/404', 'r', false, $context);
9.
10. // $stream will be a stream resource at this point regardless of
11. // the outcome of the operation
12. $body = stream_get_contents($stream);
13. $meta = stream_get_meta_data($stream);
14.
15. // $meta['wrapper_data'][0] will equal something like 'HTTP/1.0 404 Not Found'
16. // at this point, with subsequent array elements being other headers
17. $response = explode(' ', $meta['wrapper_data'][0], 3);
18. list($version, $status, $description) = $response;
19.
20. switch (substr($status, 0, 1)) {
21.     case '4':
22.     case '5':
23.         $result = false;
24.         break;
25.
26.     default:
27.         $result = true;
28. }

```


3. HTTP STREAMS WRAPPER

HTTP Authentication

The HTTP stream wrapper has no context options for HTTP authentication credentials, but you can include credentials as part of the requested URL. See the example below.

```
$response = file_get_contents('http://username:password@example.com');
```

Note that credentials are not pre-encoded; the stream wrapper handles encoding transparently when making the request.

Also, this feature supports Basic HTTP authentication, but you must handle Digest authentication manually. As such, if support for Digest authentication is a desirable feature for your project, consider using a different client library, such as one of the others discussed in later chapters of this book.

More Options

Below are other stream context options for the HTTP streams wrapper that may prove useful.

- 'user_agent' allows you to set the user agent string to use in the operation. You can also set it manually by specifying a value for the User-Agent header in the 'header' context option value.
- 'max_redirects' sets the highest number of redirects that the operation processes before assuming the application is misbehaving and terminating the request. This option is unavailable in PHP versions before 5.1.0 and uses a default value of 20.
- 'follow_location' became available in PHP 5.3.4. If you set 'max_redirects' to 1, the operation will not process redirects, but will emit an error. Setting 'follow_location' to 0 suppresses this error.
- 'timeout' sets a limit on the amount of time in seconds a read operation executes before it terminates. It defaults to the value of the `default_socket_timeout` PHP configuration setting.

All other features utilizing headers are accessible by specifying request headers in the 'header' context option and checking either `$http_response_header` or the 'wrapper_data' index of the array returned by `stream_get_meta_data()`^[8] for response headers.

[8] `stream_get_meta_data()`: http://php.net/stream_get_meta_data

Index

A

- Acceptance Tests, 159–63, 167
- Apache, 6, 12
- ASCII, 148
- authentication, 4, 6, 12, 16–19, 26, 35, 37, 52, 63, 80
 - basic, 17
 - credentials, 26, 35–37
 - digest, 17, 19, 26
 - identity, 13
 - methods, 35

B

- BrowserKit, 136–39

C

- cache, 39, 50, 86
 - content, 15, 37, 53
 - internal DNS, 39, 45
- Certificate Authority, 35, 63
 - bundle, 63–64
 - current bundle, 36, 64
- chromedriver, 138, 160–62
- Codeception, 159–63
- Composer, 36, 64, 136–37
- content encoding, 50, 85–86
- Content-Type, 10, 12, 23–24, 50, 57, 59–60, 71–72
- cookies, 4, 13, 34–35, 49, 51–52, 61–62, 73–74, 76–80, 160
 - COOKIEFILE, 34–35
 - data, 34–35, 51, 61, 76–77

- header, 13

- jar, 61
 - name, 52, 61, 76
 - objects, 77
 - store, 51
 - values, 35, 52

- CSS, 122–26, 132, 141

- CSS2, 121
- CSS3, 121
 - selectors, 4, 121–22, 124, 131, 133, 160, 163

- Ctype Extension, 149

- cURL, 27–44, 50, 52–53, 61, 63, 71, 80
 - authentication, 35
 - CURLOPT, 28–30, 33–41, 50
 - DNS caching, 39
 - Extension, 27–30, 32, 34–36, 38–40, 42, 44–46, 49, 51, 55, 75, 89
 - PHP extension, 19, 27–28, 31, 35
 - return value of, 33, 36
 - session, 28–29
 - set credentials, 35
 - target server, 40

D

- DateTime, 38, 158
- deflate, 85
 - encoding scheme, 86
- DEFLATE algorithm, 86
- denial-of-service attack, 86

INDEX

DNS, 39, 50
 caching, 39–40
 lookups, 39–40, 50
 resolution, 50
 server, 40

dnsmasq, 39–40

Document Object Model, 95

DOM

 DOMDocument, 108, 132
 DOMElement, 98–99, 105, 110
 DOMNode, 97–98, 110
 interoperability, 110, 120
 warnings, 96

DomCrawler, 133–39, 154

 component, 133

E

environments

 high load, 22
 production, 40, 91
 shared hosting, 46
 threaded, 39

G

Goutte, 136–38, 156–57

 installation instructions, 136

Guzzle, 55–56, 58, 60–64, 66, 136–37, 153, 159

 installation, 56
 manual, 59, 62–63
 request option, 63

gzinflate, 85–86

H

header

 Accept-Ranges, 16

 accessing values, 13

 associative array mapping, 58

 associative array of, 71–72

 Authorization, 17, 19

 Connection, 14, 75

 Content-Encoding, 85

 Content-Length, 84

 Content-Range, 16

 Content-Type, 24, 30, 56

 cookie request, 34–35

 custom, 53, 73

 ETag, 15

 GZIP, 86

 If-Match, 53

 If-Modified-Since, 15, 38

 If-None-Match, 15, 53

 If-Unmodified-Since, 15, 37–38, 53

 Keep-Alive, 15

 Last-Modified, 15

 Location, 36–37

 range, 16, 38, 53, 72, 147–49

 range request, 16

 referer, 14, 37

 referer request, 52

 Set-Cookie, 13, 34, 51–52, 79

 single, 71–74

 Transfer-Encoding, 84

 User-Agent, 15, 26, 38

HTTP

 HTTP/1.1, 6–7, 60, 82–83

 status codes, 7, 12, 24, 59, 71–72, 153

HttpClient, 137–38

I

IP address, 31, 39

J

jQuery, 122, 124, 129

L

libraries

guzzlehttp/promises, 64

libcurl, 27, 31, 45–46

phpQuery, 129

zlib, 86

libxml, 96–97, 108, 113–15

extension, 114

library, 118

P

Panther, 138–39

ParagonIE, 36, 64

PCRE (Perl-Compatible Regular Expression),
141, 149–50, 159

PCRE Extension, 4, 130, 141–42, 144, 146, 148

PECL, 45–46, 48, 50–55, 75

installer, 46

Perl-Compatible Regular Expression. See PCRE

persistent connections, 14–15, 84

PHP-FIG, 55

PHP-FPM, 34

phpQuery, 129–30

PHPUnit, 162

POST Requests, 10, 23, 47–48, 56, 69–70

protocol

FTP, 32

SPDY, 19

SSL, 7, 32, 36, 63

stateless, 13

TCP, 81–83

TLS, 20

PSR-7, 55

implementations, 58

interface, 56

ResponseInterface, 56, 65

StreamInterface interface, 57

PSR-18, 137

Q

query string, 8–11, 24, 30, 83–84, 135

limits, 10

parameters, 70

preformatted, 29

R

redirection, 4, 12, 14, 36, 52, 62, 80

automatic, 62

consecutive, 14

processing, 35, 62

referers, 23–24, 37, 49

regular expressions, 4, 84, 141–43, 147–48, 150

basic, 142

subpatterns, 145–46

request

body, 11, 29, 47–48, 57, 59, 70

body parameters, 70

headers, 10, 26, 59

idempotent, 10

line, 7, 12

method, 29–30, 47–48, 59, 68–69

options, 48, 50–53, 56–63, 75

sending, 83

request-response workflow, 136

INDEX

- response
 - code, 33
 - cookies, 76
 - headers, 11, 22, 26, 33, 35, 58
 - object, 50, 58, 60, 71–72, 136
 - Precondition Failed, 15
- RFC, 6–7, 10–11, 13, 15, 17, 19–20, 73, 80, 86
- robots, 166
 - exclusion, 16
 - robots.txt, 155

S

- security, 34–36, 63, 65
- Selenium, 163
- SimpleXML, 107–8, 110–11, 132
- streams wrapper, 21–22, 24, 26–27, 39, 75
- Symfony, 131–37, 139, 154
 - BrowserKit, 159
 - Console, 155
 - DomCrawler, 133, 157
 - project, 4, 131

T

- tidy, 88–93, 96
 - configurations, 90–91
 - documentation, 91
 - extension, 4, 87–90, 92–94, 96, 113
 - library, 87–88
 - output, 91
- timeout, 26, 39–40, 43, 50, 60, 75, 137

U

- Uniform Resource Identifiers (URI), 6–7
- URL
 - constant, 28

- encoding, 11, 56
- user agent, 15–16, 38, 53, 57, 80
 - sniffing, 15–16
 - spoofing, 16
 - string, 15, 26, 80
- UTF-8, 12, 33, 82, 114, 150
 - encoding, 91, 113

W

- WWW-Authenticate header, 17–19

X

- XML Parser extension, 114
- XML parsers, 4, 96, 114
- XMLReader Extension, 91, 107, 113–16, 118, 120
- XPath, 4, 100, 102, 105, 111, 122–28, 131–32
 - CSS equivalents, 121–23
 - DOMXPath, 100, 104, 132
- XPath expressions, 4, 100, 103–5, 108, 111, 121, 123, 132–33, 157, 163

Z

- Zend Framework, 67–68, 70, 72, 74, 76, 78, 128–29
 - Laminas, 67
- ZF1, 68–77, 80
- ZF2, 67–72, 74–76, 78, 80

php[architect] Books

The php[architect] series of books cover topics relevant to modern PHP programming. We offer our books in both print and digital formats. Print copy price includes free shipping to the US. Books sold digitally are available to you DRM-free in PDF, ePub, or Mobi formats for viewing on any device that supports these.

To view the complete selection of books and order a copy of your own, please visit:
<http://phparch.com/books/>.

- **Security Principles for PHP Applications**
By Eric Mann
ISBN: 978-1940111612
- **Docker for Developers, 2nd Edition**
By Chris Tankersley
ISBN: 978-1940111568 (Print edition)
- **What's Next? Professional Development Advice**
Edited by Oscar Merida
ISBN: 978-1940111513
- **Functional Programming in PHP, 2nd Edition**
By: Simon Holywell
ISBN: 978-1940111469
- **Web Security 2016**
Edited by Oscar Merida
ISBN: 978-1940111414
- **Building Exceptional Sites with WordPress & Thesis**
By Peter MacIntyre
ISBN: 978-1940111315
- **Integrating Web Services with OAuth and PHP**
By Matthew Frost
ISBN: 978-1940111261
- **Zend Framework 1 to 2 Migration Guide**
By Bart McLeod
ISBN: 978-1940111216
- **XML Parsing with PHP**
By John M. Stokes
ISBN: 978-1940111162
- **Zend PHP 5 Certification Study Guide, Third Edition**
By Davey Shafik with Ben Ramsey
ISBN: 978-1940111100
- **Mastering the SPL Library**
By Joshua Thijssen
ISBN: 978-1940111001