www.phparch.com

April 2022
Volume 21 - Issue 4

php[architect]

The Magazine For PHP Professionals

Testing The Core

Growing the PHP Core

Hack Your Home With a Pi

ALSO INSIDE

The Workshop:
Accept Testing with Codeception

Education Station:
Which License to Choose?

PHP Puzzles:
Making Some Change

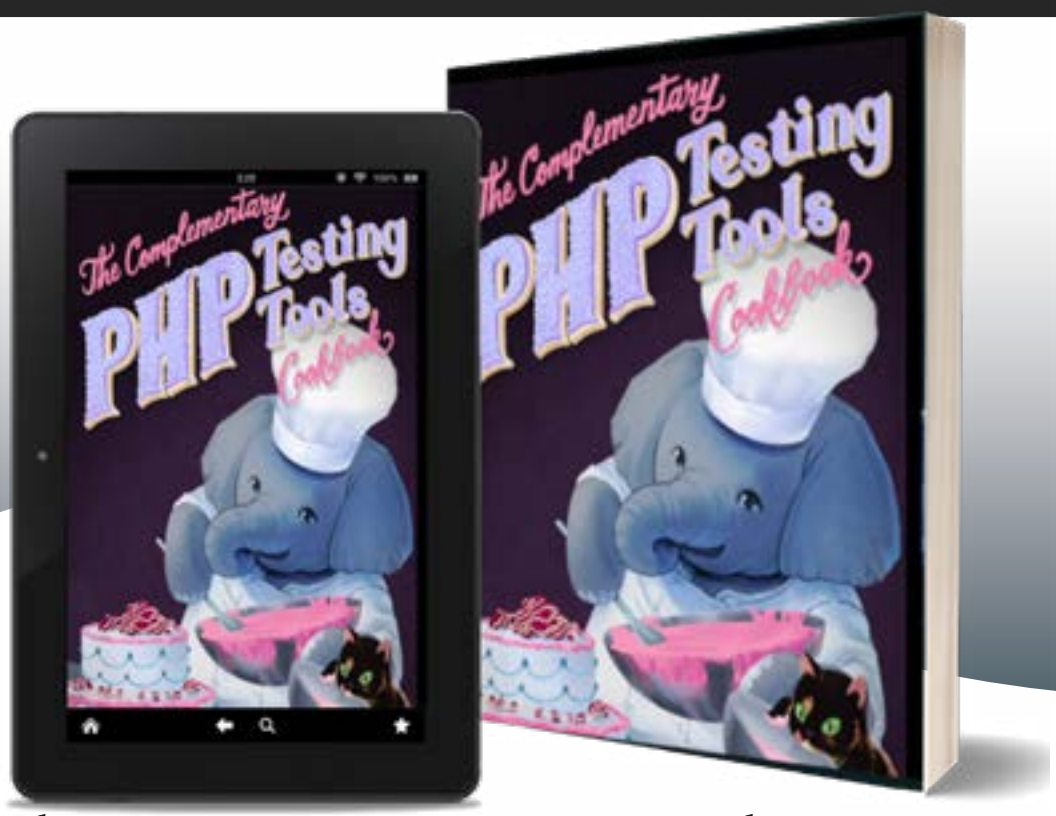
Security Corner:
Operational Security

DDD Alley:
When the New Requirement Arrives

PSR Pickup:
PSR 12 - Extending Coding Style Standards

Drupal Dab:
V9 Intro and Install

finally{}:
Tech is Taking Sides



Learn how a Grumpy Programmer approaches improving his own codebase, including all of the tools used and why.

The Complementary PHP Testing Tools Cookbook is Chris Hartjes' way to try and provide additional tools to PHP programmers who already have experience writing tests but want to improve. He believes that by learning the skills (both technical and core) surrounding testing you will be able to write tests using almost any testing framework and almost any PHP application.

Available in Print+Digital and Digital Editions.

Order Your Copy

phpa.me/grumpy-cookbook

CONTENTS

php[architect]

APRIL 2022
Volume 21 - Issue 4

2 **Impacting the World Through Tech**
Eric Van Johnson

3 **Growing the PHP Core - One Test at a Time**
Florian Engelhardt

9 **How to Hack your Home with a Raspberry Pi - Part 4**
Ken Marks

23 **Which License to Choose?**
Education Station
Chris Tankersley

28 **Operational Security**
Security Corner
Eric Mann

30 **Accept Testing with Codeception**
The Workshop
Joe Ferguson

35 **When the New Requirement Arrives**
DDD Alley
Edward Barnard

39 **Drupal 9 - Introduction and Installation**
Drupal Dab
Nicola Pignatelli

45 **Making Some Change**
PHP Puzzles
Oscar Merida

49 **New and Noteworthy**

50 **PSR 12 Extended Coding Style Standard**
PSR Pickup
Frank Wallen

52 **Tech is Taking Sides**
finally{}
Beth Tucker Long

Edited with peace and love

php[**architect**] is published twelve times a year by:

PHP Architect, LLC
9245 Twin Trails Dr #720503
San Diego, CA 92129, USA

Subscriptions
Print, digital, and corporate subscriptions are available. Visit https://www.phparch.com/magazine to subscribe or email contact@phparch.com for more information.

Advertising
To learn about advertising and receive the full prospectus, contact us at ads@phparch.com today!

Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

Impacting the World Through Tech

Eric Van Johnson

If you listen to my PHPUgly podcast, you might know that we have a policy of not mixing politics with our tech talk. That is a policy we have been breaking more and more often over the past couple of years.

From the previous presidential term and through the election, a glaring topic that started to surface more and more was that our industry and technology were playing a more significant role in politics, local and worldwide. For better or worse, it has become how people get their news, shape their opinions, and set the stage for how people are perceived. Speaking about technology and how it plays into events like the Russian invasion of Ukraine is unavoidable.

Tech companies and services also know they can no longer avoid how their products and offerings might be contributing to atrocities that they do not want to be a part of, and they have begun to take action. In this April edition of finally{}, Beth Tucker Long shares her thoughts in *Tech is Taking Sides*.

We get Part Four of Ken Marks' Raspberry Pi series, where he discusses web API and plotting data. This series has been such a fantastic read. If you haven't been keeping up, go back to the beginning of the year and read the previous three parts. Ken has been doing a terrific job explaining the journey of his real-world solution that he created using a Raspberry Pi and PHP. Also, congratulations to our second winner of the Raspberry Pi giveaway, Miguel Gallegos.

Florian Engelhardt contributed a column this month on a topic matter I had been wanting for a long time called *Growing the PHP Core One Test at a Time*. Florian walks you through the PHP core codebase and how the tests work. I couldn't wait to step through this article. He steps you through cloning the PHP codebase and through the testing workflow.

This month, we introduce a new column called Drupal Dab, contributed by Nicola Pignatelli. *Drupal 9 Introduction and Installation*—the title pretty much covers the topic. He walks us through how to install Drupal and some of the first things to do after you install it. In this month's Education Station, Chris Tankersley takes us down the path of licenses with some thoughts on choosing one for our project in his article *Which License to Choose?* He discusses the pros and cons of several open-source licenses and explains the benefits and drawbacks. In Security Corner, Eric Mann discusses *Operational Security*. He touches on what happens when disaster strikes, learning from mistakes, best practices, and the ongoing quest for security. Next to time, one of the more frustrating areas to code is money, and in this month's PHP Puzzles *Making Some Change*, Oscar Merida goes over the challenge of making change. He also shows some solutions to last month's challenge on the best ways to make change.

We've all heard the excuses for not having tests, tests are "confusing", "difficult", "takes too long to write", or are just "complicated." There's also a saying, "any tests are better than no test." In this month, The Workshop, Joe Ferguson goes over one of the easiest ways to get some basic tests in your project with his article *Accept Testing with Codeception*. I have personally been a huge fan of Edward Barnard's new DDD Alley column. This month he continues the series with *When the New Requirements Arrive*, where he talks about what you do when new requirements for a codebase are introduced and how to handle them. He touches on the solid principle, bloated classes, test boundaries, and more. Frank Wallen continues his new column, PSR Pickup, where he moves on to *PSR 12 Extended Coding Style Standard* and discusses what this PSR is and why you might want to use it in your projects.

Write For Us

If you would like to contribute, contact us, and one of our editors will be happy to help you hone your idea and turn it into a beautiful article for our magazine.

Visit https://phpa.me/write or contact our editorial team at write@phparch.com and get started!

Stay in Touch

Don't miss out on conference, book, and special announcments. Make sure you're connected with us.

- Subscribe to our list: http://phpa.me/sub-to-updates
- Twitter: @phparch
- Facebook: http://facebook.com/phparch

Download the Code Archive:

http://phpa.me/April2022_code

Growing the PHP Core - One Test at a Time

Florian Engelhardt

In September 2000, I started my vocational training at an internet agency with two teams: one doing JavaServer Pages and one doing PHP. I was assigned to the PHP team, and when presented with the language, I immediately knew that no one would ever use it. I was wrong. Today, my entire career is built on PHP. It's time to give back to the community by writing tests for the PHP core itself!

Prepare Your Machine!

Before you start writing tests for PHP, let's start with running the tests that already exist. Fetch the PHP source from GitHub and compile it to do so.

```
$ git clone git@github.com:php/php-src.git
$ cd php-src
$ ./buildconf
$ ./configure --with-zlib
$ make -j `nproc`
```

I recommend creating a fork upfront because it's easier to create a pull request when you finish your test.

If you do not have a compiler and build tools already installed on your Linux computer, install the `development-tools` group on Fedora or the `build-essential` on Debian Linux. The `./configure` command may exit with an error condition; this usually occurs when build requirements are not met, so install whatever `./configure` is missing and rerun that step. Keep in mind that you need to install the development packages — in my case, the `configure` script stated it was missing `libxml`, which was, in fact, installed. It was really missing the header files in the development package (usually named with a dev or devel suffix). Note that the `--with-zlib` argument to `./configure` is mandatory only in this case, as you will need the zlib extension (which is not built by default) for the following examples.

After your build is complete, you can find the PHP binary in `./sapi/cli/php`. Go ahead and check what you just created:

```
$ ./sapi/cli/php -v
PHP 8.2.0-dev (cli) (built: Dec 28 2021 19:43:57) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.2.0-dev, Copyright (c) Zend Technologies
```

Now that you have a freshly built and running PHP binary, you can finally run the tests included within the GitHub repository. Since PHP 7.4 tests may run in parallel, you can give the number of parallel jobs with the `-j` argument.

```
$ make TEST_PHP_ARGS=-j`nproc` test
```

```
=====================================
TEST RESULT SUMMARY
-------------------------------------
Exts skipped    :   46
Exts tested     :   26
-------------------------------------
Number of tests : 17124            11774
Tests skipped   : 5350 ( 31.2%) --------
Tests warned    :    1 (  0.0%) (  0.0%)
Tests failed    :    2 (  0.0%) (  0.0%)
Expected fail   :   28 (  0.2%) (  0.2%)
Tests passed    : 11743 ( 68.6%) ( 99.7%)
-------------------------------------
Time taken      :   40 seconds
=====================================
```

This output looks good, and it only took 40 seconds to run 11774 tests — quite fast. Only one warning and two failures, hooray! You can also see that 5350 tests have been skipped, and 28 have been expected to fail. You will learn about why this is in the next part.

What Do Tests Look Like?

Now that we know how to execute the tests, let's look at a test case. PHP test files have the file ending `.phpt` and consist of several sections, three of which are mandatory: the `TEST`, the `FILE`, and the `EXPECT` or `EXPECTF` section.

```
--TEST--
strlen() function
--FILE--
<?php
var_dump(strlen('Hello World!'));
?>
--EXPECT--
int(12)
```

The TEST Section

This section is a short description of what you are testing in a single line. It must be as short as possible, as the section is printed when running the tests. You can put additional details in the `DESCRIPTION` section.

The DESCRIPTION Section

If your test requires more than a single line to describe it adequately, you can use this section for further explanation. This section is completely ignored by the test binary.

The FILE Section

This section is the actual test case, PHP code enclosed by PHP tags. It is where you do whatever you want to test. As you have to create output to match against your expectation, you usually find `var_dump` all over the place.

The EXPECT Section

This section must exactly match the output from the executed code in the `FILE` section to pass.

The EXPECTF Section

`EXPECTF` can be used as an alternative to the `EXPECT` section and allows the usage of substitution tags you may know from the `printf` functions family:

```
--EXPECTF--
int(%d)
```

The EXTENSIONS Section

You should list all the PHP extensions that your test case depends on in this optional section. The PHP test runner will try to load those extensions, and if they are not available, skip the test. If you depend on multiple extensions, the list must be separated by a newline character.

In our case, we will list the `zlib` extension.

The SKIPIF Section

This section is optional and is parsed and executed by the PHP test runner. The test case is skipped if the resulting output starts with the word `skip`. If it starts with `xfail` the test case is run, but it is expected to fail.

The XFAIL Section

`XFAIL` is another optional section. If present, the test case is expected to fail; you don't need to echo out `xfail` in the `SKIPIF` section at all if you have this. It contains a description of why this test case is expected to fail. This feature is mainly used when the test is already finished, but the implementation isn't yet, or for upstream bugs. It usually contains a link to where a discussion about this topic can be found.

The CLEAN Section

This section exists so you can clean up after yourself. An example might be the temporary files you created during the test. Keep in mind this section's code is executed independently from the `FILES` section, so you have no access to variables declared over there. Also, this section is run regardless of the outcome of the test.

You may find more in-depth details on the file format at the PHP Quality Assurance Team Web Page[1].

What Can You Test?

Now that you know what a PHP test looks like, it is time to find something to test. Head over to Azure Pipelines[2], click on the latest scheduled run (look for the calendar icon and the string "Scheduled for"), and then click "Code Coverage" to view the code coverage report.

Figure 1.

```
543           : /* {{{ proto string zlib_get_coding_type(void)
544           :    Returns the coding type used for output compression */
545         0 : static PHP_FUNCTION(zlib_get_coding_type)
546           : {
547         0 :         if (zend_parse_parameters_none() == FAILURE) {
548         0 :                 return;
549           :         }
550         0 :         switch (ZLIBG(compression_coding)) {
551         0 :                 case PHP_ZLIB_ENCODING_GZIP:
552         0 :                         RETURN_STRINGL("gzip", sizeof("gzip") - 1);
553         0 :                 case PHP_ZLIB_ENCODING_DEFLATE:
554         0 :                         RETURN_STRINGL("deflate", sizeof("deflate") - 1);
555         0 :                 default:
556         0 :                         RETURN_FALSE;
557           :         }
558           : }
559           : /* }}} */
```

When I started looking for something to test, I found that the `zlib_get_coding_type()` function in `ext/zlib/zlib.c` was not covered at all (this was back in the PHP 7.1 branch and on the now-retired gcov.php.net[3] . See Figure 1.

The next step was to check out what this function was supposed to do in the PHP Documentation[4]. In the documentation and the code itself, I saw that this function returns the string `gzip` or `deflate` or the Boolean value `false`. The linked zlib.output_compression directive documentation[5] gave one additional bit of information: the zlib output compression feature reacts on the HTTP `Accept-Encoding` header sent with the client HTTP request.

The Big Picture

In an HTTP request, a client can indicate via the `Accept-Encoding` header that it would understand a compressed response using the specified compression algorithms. Additionally, if the PHP directive `zlib.output_compression` is enabled, PHP magically compresses the output generated by your script before sending it to the client, saving bandwidth.

1 *Web Page:* https://qa.php.net/phpt_details.php

2 *Azure Pipelines:* https://dev.azure.com/phpazuredevops/PHP/_build?definitionId=1

3 *gcov.php.net:* http://gcov.php.net/PHP_7_1/lcov_html/ext/zlib/zlib.c.gcov.php#543

4 *PHP Documentation:* https://www.php.net/zlib-get-coding-type

5 *zlib.output_compression directive documentation:* https://www.php.net/zlib.configuration#ini.zlib.output-compression

Test cases

For the test, this means there are four possible cases to check for:

- the absence of the Accept-Encoding header
- the Accept-Encoding being `gzip`
- the Accept-Encoding being `deflate`
- the Accept-Encoding being anything else

The last case is treated the same as the first case: The function is expected to return the Boolean value `false`.

Time To Write That Test!

Let's start with the first test for the case where there is no `Accept-Encoding` header set in the file `test/zlib_get_coding_type.phpt`.

Listing 1.

```
--TEST--
zlib_get_coding_type()
--EXTENSIONS--
zlib
--FILE--
<?php
ini_set('zlib.output_compression', 'Off');
var_dump(zlib_get_coding_type());
ini_set('zlib.output_compression', 'On');
var_dump(zlib_get_coding_type());
?>
--EXPECT--
bool(false)
bool(false)
```

You can run this single test via:

```
$ make test TESTS=test/zlib_get_coding_type.phpt
```

Sadly, this gives you a failed test. You can easily see why this is the case by checking the `test` directory contents where you find your `.phpt` test file and other files with various file endings. One that is particularly interesting, in this case, is the `zlib_get_coding_type.log` file:

```
---- EXPECTED OUTPUT
bool(false)
bool(false)
---- ACTUAL OUTPUT
bool(false)
Warning: ini_set(): Cannot change zlib.output_compression
  - headers already sent in test/zlib_get_coding_type.php
  on line 4
bool(false)
---- FAILED
```

Don't let these failed tests get you down if this is your first time learning about PHP internals. These failed tests are learning opportunities. For example, in this case you will learn that you cannot change the output compression setting after your script creates any form of output. For output compression to work, there is a handler that is called when we change the `zlib.output_compression` directive. This handler checks to see if any output has already been sent, and if it has, sends a warning.

Let's dive deeper. Try searching for "zlib.output_compression" in the `ext/zlib/zlib.c` source code file. You find it in the call to the `STD_PHP_INI_BOOLEAN` macro along with the pointer to the function `OnUpdate_zlib_output_compression` in which you can spot the warning you received.

Thinking of the big picture of the feature, this totally makes sense. When the script creates any form of output, PHP will create and send the HTTP response headers indicating whether the response will be compressed and, if so, what algorithm was used We can not toggle this feature anymore as there is no way of telling the client that part of the response is compressed and another part is not.

To work around this warning, you can change the `FILE` section to only create output after the calls to `ini_set`:

```
<?php
ini_set('zlib.output_compression', 'Off');
$off = zlib_get_coding_type();
ini_set('zlib.output_compression', 'On');
$on = zlib_get_coding_type();
var_dump($off);
var_dump($on);
?>
```

Rerunning the test case results in your first successful test! 🎉

It's not time to party yet; there are still other cases to cover. The next one on the list is the case with the HTTP `Accept-Encoding` header set to the string `gzip`. As a PHP developer, you know you can access and change the HTTP headers via the super global `$_SERVER`. Let's change the `FILE` and `EXPECT` section accordingly as shown in Listing 2.

Listing 2.

```
--FILE--
<?php
ini_set('zlib.output_compression', 'Off');
$off = zlib_get_coding_type();
ini_set('zlib.output_compression', 'On');
$on = zlib_get_coding_type();
$_SERVER['HTTP_ACCEPT_ENCODING'] = 'gzip';
$gzip = zlib_get_coding_type();
var_dump($off);
var_dump($on);
var_dump($gzip);
?>
--EXPECT--
bool(false)
bool(false)
string(4) "gzip"
```

Now run the test via:

```
$ make test TESTS=test/zlib_get_coding_type.phpt
```

The test failed again. Looking into the `zlib_get_coding_type.log` file, you notice that the third call to the `zlib_get_coding_type()` function returned `false`, not the expected string `gzip`. It seems like PHP is not reacting to the HTTP header that you clearly set just one line before. Did we spot a bug in PHP?

The source code tells you: Open the `ext/zlib/zlib.c` source code file and search for the variable `compression_coding` as this is the variable being evaluated in the function you are testing. You should find some matches, but there is one at the beginning of the file in the C function `php_zlib_output_encoding`, which looks like the only place where something is assigned to the variable in question.

Analyzing the source code, not understanding exactly what is going on, and finally asking for help on the PHP Community Chat[6] reveals the following: All of your userspace variables and PHP user-accessible autoglobals (`$_GET`, `$_SERVER`, …) are copy-on-write. Altering those from your script creates a copy for you to work on in userspace, effectively **making the HTTP request headers immutable to the userspace**. The PHP core continues to use the original, unaltered version.

Now that you know you can not alter or set the HTTP header from within your script to influence PHP's behavior, this might look like a sad end for your tests code coverage.

Wait, there is more! I did not tell you about another section in the PHPT file format that might come in handy now: The `ENV` section. This section is used to give environment variables to your script that are passed to the PHP process that runs your test code—meaning you can continue writing tests and that you have to create a test file per test case. Let's create a new test file for the gzip case and name it `zlib_get_coding_type_gzip.phpt`.

Listing 3.

```
--TEST--
zlib_get_coding_type() is gzip
--EXTENSIONS--
zlib
--ENV--
HTTP_ACCEPT_ENCODING=gzip
--FILE--
<?php
ini_set('zlib.output_compression', 'Off');
$off = zlib_get_coding_type();
ini_set('zlib.output_compression', 'On');
$gzip = zlib_get_coding_type();
var_dump($on);
var_dump($gzip);
?>
--EXPECT--
bool(false)
string(4) "gzip"
```

6 *PHP Community Chat: https://phpc.chat*

Listing 4.

```
--TEST--
zlib_get_coding_type() is gzip
--EXTENSIONS--
zlib
--ENV--
HTTP_ACCEPT_ENCODING=gzip
--FILE--
<?php
ini_set('zlib.output_compression', 'Off');
$off = zlib_get_coding_type();
ini_set('zlib.output_compression', 'On');
$gzip = zlib_get_coding_type();
ini_set('zlib.output_compression', 'Off');
var_dump($off);
var_dump($gzip);
?>
--EXPECT--
bool(false)
string(4) "gzip"
```

Run the test and see it fails once more. You might have expected this by now. 😜

Let's see what happened by looking into the `zlib_get_coding_type_gzip.log` file.

```
---- EXPECTED OUTPUT
bool(false)
string(4) "gzip"
---- ACTUAL OUTPUT
□1□0
    □□□B□P□Tx□8`5□□□□□□□s□7a□ze□□B+□□□□□,□□□□^□
---- FAILED
```

Wow, this looks like some binary garbage, and yeah, this does not match the expected output.

Look at your test case. You told PHP that you accept gzip encoding and you turned on output compression. PHP is basically just doing what we told it to do. The binary garbage you see here is gzip encoded data. This means you succeeded in activating gzip output compression, but forgot to turn it off before dumping out the two variables. Now, all that's left is to add another call to deactivate output compression again so the final test looks similar to Listing 4.

Rerun this test, and it finally passes! 🎊 🎉

Now you're left writing one test case for the `Accept-Encoding` header being set to `deflate` and another test case for the `Accept-Encoding` header being set to an invalid string (basically something that is not `gzip` or `deflate`). From this point on, finishing the test cases is just diligence, and if you like to cheat, you can find the tests in the `ext/zlib/tests/` directory named:

- `zlib_get_coding_type_basic.phpt`
- `zlib_get_coding_type_br.phpt`
- `zlib_get_coding_type_deflate.phpt`
- `zlib_get_coding_type_gzip.phpt`

Figure 2.

```
541           : /* {{{ proto string zlib_get_coding_type(void)
542           :    Returns the coding type used for output compression */
543         8 : static PHP_FUNCTION(zlib_get_coding_type)
544           : {
545         8 :         if (zend_parse_parameters_none() == FAILURE) {
546         0 :                 return;
547           :         }
548         8 :         switch (ZLIBG(compression_coding)) {
549         1 :                 case PHP_ZLIB_ENCODING_GZIP:
550         2 :                         RETURN_STRINGL("gzip", sizeof("gzip") - 1);
551         1 :                 case PHP_ZLIB_ENCODING_DEFLATE:
552         2 :                         RETURN_STRINGL("deflate", sizeof("deflate") - 1);
553         6 :                 default:
554         6 :                         RETURN_FALSE;
555           :         }
556           : }
557           : /* }}} */
```

Collect Some Evidence!

Now that you have tests for all four cases, you could hope you covered that function with tests or (a way better option if you ask me), you could continue and create a code coverage report! For the report to work, you need to install the `lcov` tool via your Linux package manager (`sudo dnf install lcov` for Fedora or `sudo apt-get install lcov` for Debian). PHP was built with `gcov` disabled, so we need to run configure again, with enabled `gcov` and recompile.

```
$ make clean
$ CCACHE_DISABLE=1 ./configure --with-zlib --enable-gcov
$ make -j `nproc`
```

This code compiles the PHP binary with enabled code coverage generation. Lcov will be used to create an HTML code coverage report afterward. To create your code coverage report, you need to rerun the tests.

```
$ make test TESTS=test/zlib_get_coding_type.phpt
```

While running the tests, `gcov` generates coverage files with a `gcda` file ending for every source code file. To generate the HTML code coverage report, run the following:

```
$ make lcov
```

You may find the resulting HTML code coverage report in `lcov_html/index.html`.

It looks like we covered all four cases as shown in Figure 2.

What's in It for You?

PHP becomes more stable and reliable with every test you write, as functions may not suddenly change behavior between releases.

Writing tests for PHP gives you a deeper understanding of how your favorite language works internally. In fact, by reading up to this point, you may have learned that HTTP request headers are immutable to userspace, that there is such a thing as userspace, and that there are handler functions called to check what you are doing when you want to change ini settings at runtime.

Also, knowing the PHPT file format gives you other benefits as well: PHPUnit[7] not only supports the PHPT file format and can execute those tests, part of PHPUnit is tested with PHPT test cases. This skill led me to become a PHPUnit contributor: Writing PHPUnit tests in the PHPT file format.

Shout Out!

I would like to thank everyone involved in the PHP TestFest[8], especially Ben Ramsey[9] and everyone else involved in the organization of the 2017 edition. This was what made me write tests for PHP and made me not only a contributor to PHP but also a PHPUnit contributor. In the long run, this brought me my first ElePHPant 🐘!

Florian is a father of five, developer, architect, tech lead, geek and still sometimes finds time to contribute to open source. He started writing software with PHP in September 2000 and will try to convince you to use vim. @realFlowControl

7 *PHPUnit:* https://phpunit.de/

8 *PHP TestFest:* https://phptestfest.org/start/#whos-behind-this

9 *Ben Ramsey:* https://twitter.com/ramsey

How to Hack your Home with a Raspberry Pi - Part 4

Ken Marks

Welcome back to another installment of How to Hack your Home with a Raspberry Pi*. At the end of this article, you will have an accelerometer data plotting application running on your Raspberry Pi. You will need to start from the beginning of the series if you want to build this project yourself.

Introduction

Here's what we've covered so far in parts 1 thru 3:

- Part 1:
  - My motivation for this project
  - Components needed
  - Installing the OS on the Pi
  - Configuring the Pi
- Part 2:
  - Updating the software packages on your Pi
  - Installing and testing the LAMP Stack
- Part 3:
  - Connecting the accelerometer
  - Creating the database for storing accelerometer data
  - Compiling a C/C++ program that reads accelerometer data and logs it to the database
  - Creating a Unix Service to automatically start and stop logging

In this installment we will:

- build a web service that uses the logged accelerometer data
- and build an application that plots the accelerometer data

Power Up Your Pi and Remotely Log in

Ensuring your accelerometer is properly connected to your Raspberry Pi (see part 3), connect the power supply to your Pi and plug it into A/C mains.

Ssh Into Your Pi

After a few minutes of letting the Pi power-up, bring up a terminal window on your computer. Next, SSH into your Pi with the user `pi` by typing the following command into the terminal:

```
ssh pi@raspberrypi.local
```

Enter your password. You should now be logged in to your Raspberry Pi.

Update the Packages

I'll first check to see if any software packages need upgrading on the Pi using the *apt* package manager and run the following two commands in the terminal window:

```
sudo apt update
sudo apt upgrade
```

Building a Web Service

We ultimately want to build a web application that plots the accelerometer data in real time. So first, let's create a simple REST service with a single endpoint that provides the following two mechanisms for reading the accelerometer data:

- Getting the newest accelerometer data and its `id`
- Getting all the accelerometer data starting with the next newest row of data after the `id` sent, all the way up to the newest data along with the newest accelerometer data's `id`.

> 💡 *What is REST? REST[1] stands for REpresentational State Transfer. If this acronym isn't confusing enough, we have many alternative names for it such as RESTful service, web API, or web service. At the great risk of giving an oversimplified definition, REST is a means of transferring data (typically defined as a noun in the URI) using HTTP verbs representing the intent of what we are doing with the data. Going forward, I will refer to this as a web service. We use web services to expose an API for accessing a CRUD object (another acronym meaning Create, Read, Update, and Delete, which is a code wrapper for accessing our database tables).*

1 REST: https://w.wiki/7iv

> *We'll create a CRUD wrapper which I'll describe in more detail below. Instead of having separate URIs for each function of our CRUD operation, a web service allows us to have a single URI, and we use the verbs that associate with each CRUD operation. See Table 1.*

Since all we need from our web service is to retrieve data, we will only use the *HTTP* verb `GET`. When a client makes an *HTTP GET* request to our service, we'll return a JSON[2] payload containing the accelerometer data, making it really flexible for any programming language to interact with this service. For example, I could create an application entirely in PHP using Guzzle[3] or in JavaScript using AJAX to interact with the web service.

Accelerometer Data Web Service Api

Table 1.

HTTP Verb	CRUD Action
POST	Create record
GET	Read record(s)
PUT	Update record
DELETE	Delete record

We won't require any authentication to keep it simple, and we won't worry about versioning our service either. As mentioned, there is just one endpoint, and we will create it at the following location to interact with our web service:

```
http://raspberrypi.local/accelerometerservice/
```

The first time we call our service, we want to get the latest accelerometer data, so we will send an *HTTP GET* request without any parameters as shown in Table 2 row 1.

We want the *JSON* data we get back to contain the latest accelerometer reading, its timestamp, and its ID. See Listing 1.

Then we can plot this data point in the plotting application we will create using the timestamp.

Remember, our accelerometer logging service is saving data every 100 milliseconds. Therefore, we will use the `lastMeasurementId` as an input parameter the next time we request data from the web service to get all the data points since the above request.

Table 2.

HTTP Verb	Endpoint	Parameter
GET	http://raspberrypi.local/accelerometerservice/	[none]
GET	http://raspberrypi.local/accelerometerservice/	lastMeasurementId

2 *JSON:* https://www.json.org/json-en.html

3 *Guzzle:* https://docs.guzzlephp.org/en/stable/

Listing 1.

```
1. "accelerationData": {
2.   "accelerationMeasurements": [
3.     {
4.       "axis": {
5.         "X": "0.199219",
6.         "Y": "-0.182617",
7.         "Z": "0.943359"
8.       },
9.       "dateTime": "2022-03-02 15:00:46.631"
10.    }
11.  ],
12.  "lastMeasurementId": "171695"
13. }
```

For our next call to our service, we will send an *HTTP GET* request (Table 2 row 2) with the `lastMeasurementId` as a parameter to get all the data since our last request.

We want the *JSON* data we get back to contain all the accelerometer data since the last request and the newest reading's ID as shown in Listing 2.

If this request is exactly 10 seconds since the last request, we should have 100 measurements in the *JSON* payload.

Creating the Accelerometer Data Web Service

I have created a PHP web service containing the endpoint script and a CRUD wrapper class to retrieve the acceleration data from the database and convert it into a *JSON* payload to be sent to the requesting client.

Listing 2.

```
1. "accelerationData": {
2.   "accelerationMeasurements": [
3.     {
4.       "axis": {
5.         "X": "0.203125",
6.         "Y": "-0.177734",
7.         "Z": "0.947266"
8.       },
9.       "dateTime": "2022-03-02 15:00:46.746"
10.    },
11.    ...
12.    {
13.      "axis": {
14.        "X": "0.200195",
15.        "Y": "-0.185547",
16.        "Z": "0.948242"
17.      },
18.      "dateTime": "2022-03-02 15:00:56.625"
19.    }
20.  ],
21.  "lastMeasurementId": "171795"
22. }
```

Download the Accelerometer Service Scripts

In the terminal window, within your `Downloads` folder, download the `accelerometerservice.zip`[4] file to your Pi using `wget`:

```
cd ~/Downloads/
wget https://www.phparch.com/downloads/accelerometerservice.zip
```

In the terminal window, unzip the `accelerometerservice.zip` file and move the `accelerometerservice` folder to the `/var/www/html/` folder:

```
unzip accelerometerservice.zip
sudo mv accelerometerservice/ /var/www/html/
```

The web service contains the scripts shown in Table 3.

And our web service endpoint is now located at: `http://raspberrypi.local/accelerometerservice/`

accelerometerdata.php

First, let's take a look at `AccelerometerData.php` in Listing 3.

This class is for fetching data from a database table into an object. You can see that the properties in this class match the fields in the **accelerometer_data** table in the **AccelerometerData** database. Each object of this class will represent a row from a query into the table. I've added magic getters and setters and a `__toString()` method for completeness, but we'll just be using the magic getter.

accelerometerdatamanager.php

Listing 4 is our CRUD wrapper the `AcelerometerDataManager` class. This class has two `public` methods:

- `readLatest()` retrieves the newest accelerometer data, including its `id`

Listing 3.

```
1. class AccelerometerData
2. {
3.     private $id;
4.     private $created;
5.     private $axis_x;
6.     private $axis_y;
7.     private $axis_z;
8.
9.     // Get/Set
10.    public function __get($ivar)
11.    {
12.        return $this->$ivar;
13.    }
14.
15.    public function __set($ivar, $value)
16.    {
17.        $this->$ivar = $value;
18.    }
19.
20.    // Serialize
21.    public function __toString()
22.    {
23.        $format =
24.            "<hr/>Id: %s<br/>Created: "
25.            . "%s<br/>X: %s<br/>Y: "
26.            . "%s<br/>Z: %s<hr/>";
27.
28.        return sprintf(
29.            $format,
30.            $this->__get('id'),
31.            $this->__get('created'),
32.            $this->__get('axis_x'),
33.            $this->__get('axis_y'),
34.            $this->__get('axis_z')
35.        );
36.    }
37. }
```

Table 3.

File Name	Description
AccelerometerData.php	Class wrapper for accelerometer_data table data
AccelerometerDataManager.php	CRUD manager wrapper for retrieving data from the accelerometer_data table and packaging it up into a JSON payload
index.php	Web service entrypoint

4 accelerometerservice.zip: https://www.phparch.com/downloads/accelerometerservice.zip

Listing 4.

```
require_once('AccelerometerData.php');

class AccelerometerDataManager
{
    const HOST = "localhost";
    const DB = "AccelerometerData";
    const USER = "accelerometer";
    const PW = "accelerometer";

    public function readLatest()
    { ... }

    public function readFromIdToLatest($id)
    { ... }

    private function
    getJsonEncodedDataFromAccelDataObjects(
        $accelDataObjects
    )
    { ... }
}
```

- `readFromIdToLatest($id)` retrieves all the accelerometer data since the `id` (passed in as a parameter) and the newest `id`. If the `id` is not found in the table (due to it being deleted because it's older than 60 seconds), return all 60 rows of the accelerometer data.

The class also has one `private` utility method:

- `getJsonEncodedDataFromAccelDataObjects($accelDataObjects)` that we will use to create a nested associative array from the accelerometer data and convert it into a *JSON* payload.

Let's take a look at the `getJsonEncodedDataFromAccelDataObjects()` (Listing 5) method first.

The parameter,`$accelDataObjects`, is an array of one or more `AccelerometerData` objects fetched from the **accelerometer_data** table.

The `foreach()` loop builds a nested associative array (called `$measurements`) containing the accelerometer data from the array of `AccelerometerData` objects.

After processing all of the accelerometer data, we will continue to build the nested associative array by creating an array called `$accelerationData` with a key of `accelerationData`. The key contains another array with two key/value pairs: `accelerationMeasurements` containing the `$measurements` array as a value, and `lastMeasurementId` containing the value of the last ID processed from the `AccelerometerData` object array(which will be the newest reading).

Listing 5.

```
private function
    getJsonEncodedDataFromAccelDataObjects($accelDataObjects)
{
    $measurements = array();

    foreach ($accelDataObjects as $accel)
    {
        $measurements[] =
            array(
                'axis' =>
                    array(
                        'X' => $accel->axis_x,
                        'Y' => $accel->axis_y,
                        'Z' => $accel->axis_z
                    ),
                'dateTime' => $accel->created
            );
    }

    $accelerationData =
        array(
            'accelerationData' =>
                array(
                    'accelerationMeasurements' => $measurements,
                    'lastMeasurementId' => $accel->id
                )
        );

    return json_encode($accelerationData, JSON_PRETTY_PRINT);
}
```

Finally, we call `json_encode()` to convert the completed associative array `$accelerationData` into a pretty printed *JSON* payload and return it to the calling method.

Let's take a look at the `readLatest()` method in Listing 6, which will return the newest accelerometer data reading along with its primary key ID.

Using PDO, we create a connection to the **AccelerometerData** database and we will handle exceptions. Next, we will query the newest row of data in the **accelerometer_data** table and fetch the results into an array of `AccelerometerData` objects — there should only be one object in the array. If

something goes wrong with the query, we will return 0 to the calling function. We pass this `AccelerometerData` object array to the `getJsonEncodedDataFromAccelDataObjects()` method and return the *JSON* payload to the calling function.

Finally, let's look at the `readFromIdToLatest($id)` method in Listing 7 which will return all the accelerometer data readings since the reading specified by the `$id` parameter.

This method takes an ID corresponding to a row in the **accelerometer_data** table. Using PDO, we create a connection to the **AccelerometerData** database, and we will handle exceptions. Next, we will query all the newest rows of data in the **accelerometer_data** table greater than the `$id` passed in and fetch the results into an array of `AccelerometerData` objects. Notice we are preparing and binding our incoming `$id` to `:id` to prevent SQL injection. If something goes wrong with the query, we will call the `readLatest()` method to return the newest accelerometer data reading. Finally, we pass this `AccelerometerData` object array to the `getJsonEncodedDataFromAccelDataObjects()` method and return the *JSON* payload to the calling function.

Listing 6.

```
1. public function readLatest()
2. {
3.   $retVal = NULL;
4.
5.   $db = new PDO("mysql:host=" . self::HOST . ";dbname="
6.       . self::DB, self::USER, self::PW);
7.   $db->setAttribute(PDO::ATTR_ERRMODE,
8.       PDO::ERRMODE_EXCEPTION);
9.
10.  // Get the latest measurement
11.  $sql = "SELECT id, created, axis_x, axis_y, axis_z "
12.      . "FROM accelerometer_data "
13.      . "ORDER BY id DESC LIMIT 1";
14.
15.  try
16.  {
17.    $query = $db->prepare($sql);
18.    $query->execute();
19.
20.    $results = $query->fetchAll(PDO::FETCH_CLASS,
21.      "AccelerometerData");
22.
23.    if (is_array($results) && count($results) == 1)
24.    {
25.      return $this->
26.        getJsonEncodedDataFromAccelDataObjects($results);
27.    }
28.    else
29.    {
30.      $retVal = 0;
31.      $retVal = json_encode($retVal, JSON_PRETTY_PRINT);
32.    }
33.  }
34.  catch(Exception $ex)
35.  {
36.    echo "{$ex->getMessage()}<br/>\n";
37.  }
38.
39.  return $retVal;
40. }
```

index.php

`index.php` (Listing 8) serves as the entry to our web service endpoint.

First we grab the HTTP REQUEST method from `$_SERVER['REQUEST_METHOD']`. Next, we instantiate an `AccelerometerDataManager` object. Then we will *switch* on `$httpVerb`. We will only process *HTTP GET* requests and throw an Unsupported HTTP request exception for anything else.

Listing 7.

```
1. public function readFromIdToLatest($id)
2. {
3.   $retVal = NULL;
4.
5.   $db = new PDO("mysql:host=" . self::HOST . ";dbname="
6.     . self::DB, self::USER, self::PW);
7.   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
8.
9.   // Get the 'created' date from the given id
10.  $sql = "SELECT id, created, axis_x, axis_y, axis_z "
11.    . "FROM accelerometer_data WHERE id > :id "
12.    . "ORDER BY id";
13.
14.  try
15.  {
16.    $query = $db->prepare($sql);
17.    $query->bindParam(":id", $id, PDO::PARAM_INT);
18.    $query->execute();
19.
20.    $results = $query->fetchAll(
21.        PDO::FETCH_CLASS,
22.        "AccelerometerData"
23.      );
24.
25.    if (is_array($results) && count($results) >= 1)
26.    {
27.      return
28.        $this->getJsonEncodedDataFromAccelDataObjects($results);
29.    }
30.    else
31.    {
32.      // Just get the latest reading
33.      $retVal = $this->readLatest();
34.    }
35.  }
36.  catch(Exception $ex)
37.  {
38.    echo "{$ex->getMessage()}<br/>\n";
39.  }
40.
41.  return $retVal;
42. }
```

Once we've determined the request method is an *HTTP GET*,

we will send an HTTP header with the content type of `application/json`, so the client's browser knows to expect a *JSON* payload. If a query parameter named `lastMeasurementId` is passed in we will call the `readFromIdToLatest()` method passing in the query parameter contained in `$_GET['lastMeasurementId']`. Otherwise, we will call the `readLatest()` method. Both of these methods will return a *JSON* payload which is echoed out to the client.

Listing 8.

```
1. require_once("AccelerometerDataManager.php");
2.
3. $httpVerb = $_SERVER['REQUEST_METHOD']; // POST, GET, PUT, DELETE, ...
4.
5. $accelerometerDataManager = new AccelerometerDataManager();
6.
7. switch ($httpVerb)
8. {
9.     case "GET":
10.         // Read
11.         header("Content-Type: application/json");
12.         if (isset($_GET['lastMeasurementId'])) // Read (by lastMeasurementId)
13.         {
14.             echo $accelerometerDataManager->readFromIdToLatest($_GET['lastMeasurementId']);
15.         }
16.         else
17.         {
18.             echo $accelerometerDataManager->readLatest();
19.         }
20.         break;
21.
22.     default:
23.         throw new Exception("Unsupported HTTP request");
24.         break;
25. }
```

Testing Our Web Service

We have several ways we can test our web service to see if we're getting the expected *JSON* payload:

- Using our browser
- Running `curl` from a terminal window on our local computer — easier if on a UNIX based OS
- Running the **Postman** application - a popular application for testing web services

I'll show you how to test your web service using a browser and by running `curl` in your local computer's terminal window (since I'm running macOS).

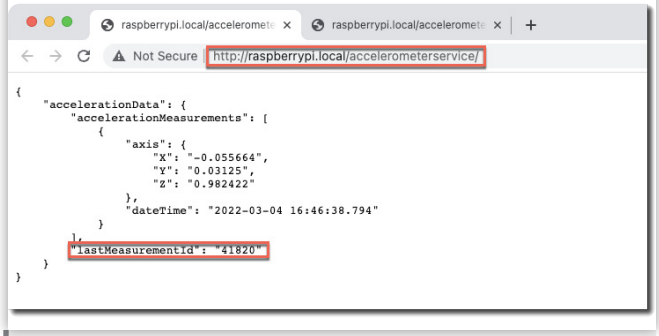
> 💡 *When testing a web service using the browser, we can really only test HTTP GET and POST request methods (unless we write an application using Guzzle). We can test GET methods just using the URL entry and POST by creating a form. To test the other HTTP verbs, you should use `curl` or, better yet, an application like `Postman`.*

Bring up a browser tab on your local computer and navigate to `http://raspberrypi.local/accelerometerservice/`. You should see a *JSON* payload that returns the newest accelerometer data and looks something like Figure 1.

Notice the `lastMeasurementId` and its value. We will use this as a query parameter in our next request to query all the accelerometer data since this request.

> 💡 *NOTE: When we request all the data since this last request, keep in mind that the `accelerometerdatalogging.service` is only storing 60 seconds worth of data. So if you request all the data since the `lastMeasurementId` and more than 60 seconds have passed, you will retrieve the latest 60 seconds worth of accelerometer measurement data. This web service is designed to be used in a real-time application. The plotting application we are building will request data from this service every one second.*

Figure 1.



Assuming less than 60 seconds have transpired, bring up another tab in your browser and navigate to `http://raspberrypi.local/accelerometerservice/?lastMeasurementId=41820`. You should see a *JSON* payload that returns all the newest accelerometer data since the `lastMeasurementId` and looks something like Figures 2 and 3.

Now, I'll show you how to use `curl` to test your web service if you have it installed as a command-line utility. Open up a terminal window on your local computer and type the following command:

```
curl --location --request \
GET 'http://raspberrypi.local/accelerometerservice'
```

Figure 2. JSON payload part 1

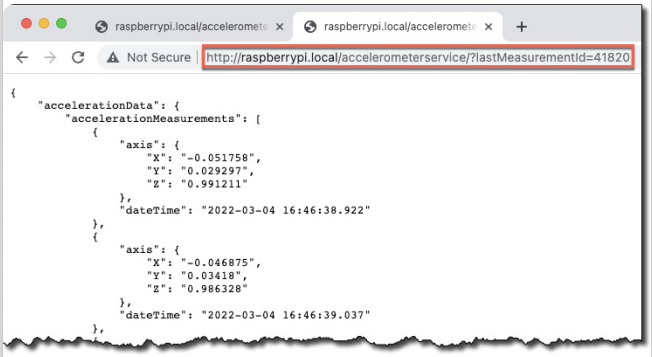
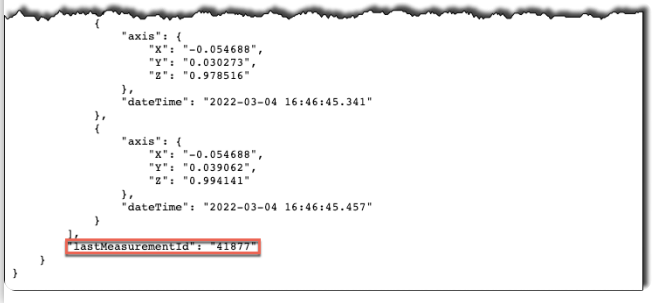


Figure 3. JSON payload part 2



In Figure 4, you should see the *JSON* payload returned. If you like, you can repeat this command with the query parameter added onto the end of the URL to get all the newest measurement data since the `lastMeasurementId`.

Building a Plotting Application

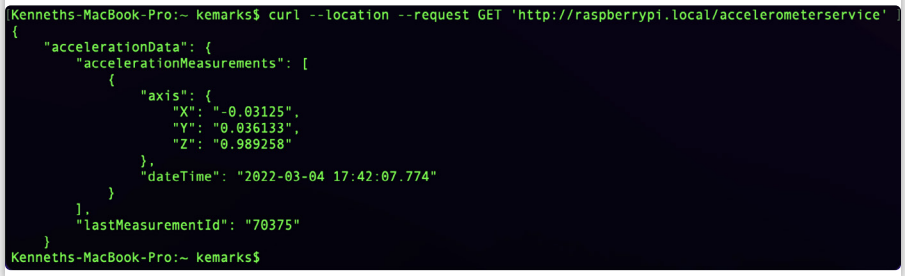
Now that we know our accelerometer data web service is working, we will build a client-side web application using JavaScript that makes an AJAX call to the web service to get the latest accelerometer entries every second. We will use *Smoothie Charts*[5] (a JavaScript charting library) to plot the accelerometer data in real time.

Download the Accelerometer Plotting Application Scripts

In the terminal window, within your `Downloads` folder, download the `accelerometer.zip`[6] file to your Pi using `wget`:

```
cd ~/Downloads/
wget https://www.phparch.com/downloads/accelerometer.zip
```

Figure 4.



In the terminal window, unzip the `accelerometer.zip` file and move the `accelerometer` folder to the `/var/www/html/` folder:

```
unzip accelerometer.zip
sudo mv accelerometer/ /var/www/html/
```

Table 4.

File Name	Description
smoothie.js	Smoothie Charts library script
index.html	Main page that displays the canvas containing the real-time plot of accelerometer data
js_get_server_name.php	PHP script for returning the name of the server to an AJAX call for proper URL construction
acceleration.js	Contains functions for getting the latest accelerometer data from the web service and plotting it to a canvas using Smoothie Charts

The plotting application contains the following scripts shown in Table 4.

Our accelerometer plotting application is now located at: `http://raspberrypi.local/accelerometer/`.

I'm not going to go through the `smoothie.js` library, but if you want to know more, a ten-minute tutorial[7] shows you how easy it is to use. And a configuration tool called *Smoothie Charts Builder*[8] allows you to set various options and generates the HTML and JavaScript you need for these options.

5 *Smoothie Charts:* http://smoothiecharts.org

6 *accelerometer.zip:* https://www.phparch.com/downloads/accelerometer.zip

7 *ten-minute tutorial:* http://smoothiecharts.org/tutorial.html

8 *Smoothie Charts Builder:* http://smoothiecharts.org/builder/

However, I want to walk through the rest of the scripts with you so you understand how our plotting application works and integrates with our web service.

> *NOTE: Smoothie Charts has many options not covered in the Smoothie Charts Builder. You will need to peruse the well-documented* `smoothie.js` *library to find them.*

index.html

I will again use the Bootstrap[9] CSS framework for our *Accelerometer Data* page. See Listing Listing 9.

We're starting out with a typical boilerplate *Bootstrap* web page. The things I want you to note are we need a canvas:

```
<canvas id="acceleration_data_plot_canvas"
style="width:100%; height:100px"></canvas>
```

Our canvas will be a fixed height of 100 pixels and responsive to the width of the canvas.

Next, we need to include *JQuery* because we will be making some *AJAX* calls to use our web service:

```
<!-- Compiled and minified JQuery -->
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
 integrity="sha256-/xUj+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
 crossorigin="anonymous"></script>
```

Then we need to include our `smoothie.js` library and our `acceleration.js` script containing our functions:

```
<!-- Smoothie Charts and Accelerometer Data Plotting Code -->
<script type="text/javascript" src="smoothie.js"></script>
<script type="text/javascript" src="acceleration.js"></script>
```

And lastly we need to call our `plotAccelerometerData()` function:

```
<script type="text/javascript">
  plotAccelerometerData();
</script>
```

js_get_server_name.php

This script will return the name of the server as *JSON* to an AJAX call from the `acceleration.js` script and will be used to construct the URL to the web service endpoint:

Listing 9.

```
17.     <div class="card">
18.       <div class="card-body">
19.           <div class="alert alert-success" role="alert">
20.             <h3>Accelerometer Data</h3>
21.           </div>
22.       </div>
23.     </div>
24.     <div class="card">
25.       <div class="card-body">
26.           <canvas id="acceleration_data_plot_canvas" style="width:100%; height:100px"></canvas>
27.       </div>
28.     </div>
29.     <!-- Bootstrap 5 compiled and minified bundled JavaScript -->
30.     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
31.             integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
32.             crossorigin="anonymous"></script>
33.
34.     <!-- Compiled and minified JQuery -->
35.     <script src="https://code.jquery.com/jquery-3.6.0.min.js"
36.             integrity="sha256-/xUj+30JU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
37.             crossorigin="anonymous"></script>
38.
39.     <!-- Smoothie Charts and Accelerometer Data Plotting Code -->
40.     <script type="text/javascript" src="smoothie.js"></script>
41.     <script type="text/javascript" src="acceleration.js"></script>
42.
43.     <script type="text/javascript">
44.       plotAccelerometerData();
45.     </script>
...
```

9 *Bootstrap: http://getbootstrap.com*

Listing 10.

```
let Server =
{
    url: ""
}

let AccelerometerData =
{
    lastId: 0
}
```

Listing 11.

```
function getLatestAccelerometerData(
           axis_x,
           axis_y,
           axis_z)
{ ... }

function plotAccelerometerData()
{ ... }
```

Listing 12.

```
function plotAccelerometerData () {
  $.getJSON('js_get_server_name.php',
    function (data) {
      Server.url = 'http:\/\/'
        + Server.server_name
        + '/accelerometerservice/';
    });
...
}
```

```
<?php
    header('Content-type: application/json');
    $server_name = $_SERVER['SERVER_NAME'];
    print json_encode(array('server_name' => $server_name));
```

acceleration.js

This script has functions for getting the latest accelerometer data from the web service and plotting it to a canvas using **Smoothie Charts**.

First, we will create a couple of properties. The first property is for holding the URL to our web service endpoint used in AJAX calls to the service. The second property will hold the ID of the last read accelerometer data. See Listing 10.

In Listing 11 we will have two functions: `getLatestAccelerometerData()` and `plotAccelerometerData()`. Let's look at the `plotAccelerometerData()` function first, as it will show us how to set up and use the **Smoothie Charts** library. `plotAccelerometerData()` is the entry point that is called from `index.html`.

First, we need to construct a proper URL of our web service endpoint (Listing 12), so we will use `jQuery.getJSON()` to get our JSON encoded server name using an AJAX HTTP GET request and set the `Server.url` property to point to our web service with the endpoint URL: `http://raspberrypi.local/accelerometer/`.

Next, we need to specify the plot lines for each axis of accelerometer data. **Smoothie Charts** requires us to create a `TimeSeries` object for each plotline:

```
function plotAccelerometerData()
{
        ...
        let x = new TimeSeries();
    let y = new TimeSeries();
    let z = new TimeSeries();
        ...
}
```

Then we will call `setInterval()`, which will call the `getLatestAccelerometerData()` function every one second, passing in the time series objects for each axis. `getLatestAccelerometerData()` will retrieve one second's worth of accelerometer data and append it to the time series objects. See Listing 13.

In Listing 14 we will instantiate our `SmoothieCart` object.

We will set our chart to be responsive, so when we resize the width of our canvas, the chart will adjust accordingly. We also set the `grid` parameters and `labels`. You can experiment with some of these settings using the *Smoothie Charts*

Listing 13.

```
function plotAccelerometerData () {
...
  setInterval(function () {
    getLatestAccelerometerData(x, y, z);
  }, 1000);
...
}
```

Listing 14.

```
function plotAccelerometerData () {
...
  let smoothie = new SmoothieChart(
    {
      responsive: true,
      grid:
        {
          strokeStyle: 'rgb(125, 0, 0)',
          fillStyle: 'rgb(60, 0, 0)',
          lineWidth: 1,
          millisPerLine: 250,
          verticalSections: 6
        },
      labels: {fillStyle: 'rgb(255, 255, 0)'}
    });
...
}
```

Builder[10] or by looking at the options in the function header documentation for the `smoothie` constructor in `smoothie.js`.

Then we will add the three `TimeSeries` objects to our `SmoothieChart` object and set the `strokeStyle`, `fillStyle`, and `lineWidth` parameters for each axis by calling the `SmoothieChart.addTimeSeries()` method:

```
function plotAccelerometerData()
{
    ...
    smoothie.addTimeSeries(x, {
        strokeStyle: 'rgb(0, 255, 0)',
        fillStyle: 'rgba(0, 255, 0, 0.3)',
        lineWidth: 3 });
    smoothie.addTimeSeries(y, {
        strokeStyle: 'rgb(255, 0, 255)',
        fillStyle: 'rgba(255, 0, 255, 0.3)',
        lineWidth: 3 });
    smoothie.addTimeSeries(z, {
        strokeStyle: 'rgb(0, 0, 255)',
        fillStyle: 'rgba(0, 0, 255, 0.3)',
        lineWidth: 3 });
        ...
}
```

We'll set the X-axis green, the Y-axis magenta, and the Z-axis blue.

Finally, we will call the `SmoothieChart.streamTo()` method to stream the accelerometer data to the canvas we identified with the element ID `acceleration_data_plot_canvas` with a delay of one second to avoid jitter in the display:

```
function plotAccelerometerData()
{
    ...
    smoothie.streamTo(document.getElementById(
        "acceleration_data_plot_canvas"), 1000);
    ...
}
```

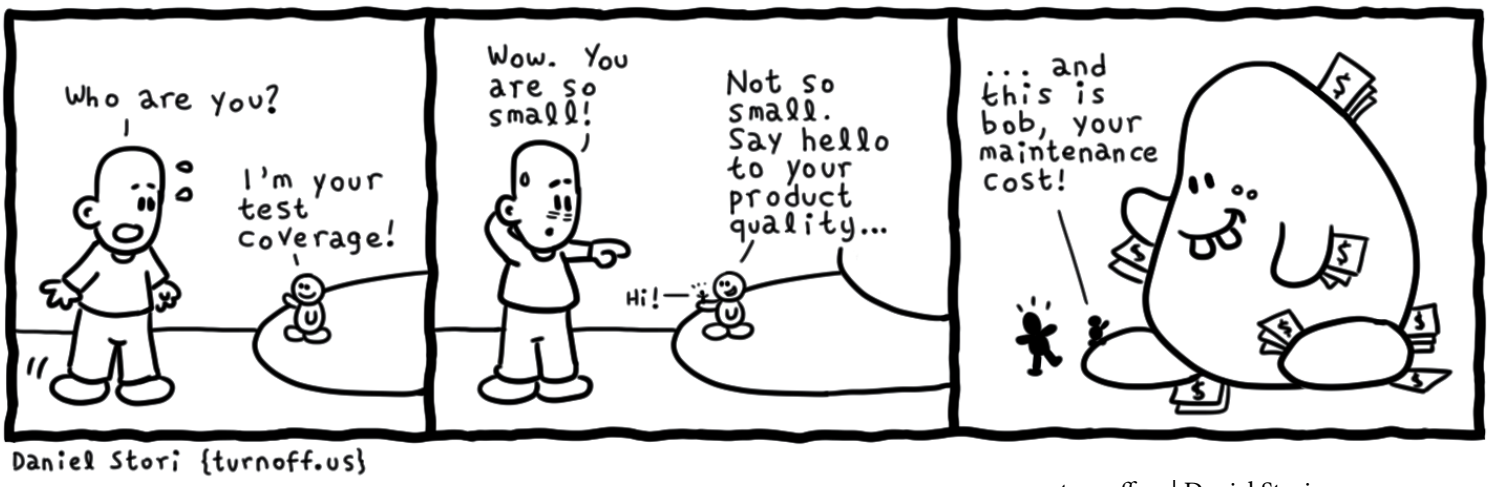
Listing 15 shows the `getLatestAccelerometerData()` function, which is responsible for getting the accelerometer data from the web service and adding it to the Smoothie Charts `TimeSeries` objects.

Listing 15.

```
function getLatestAccelerometerData(axis_x, axis_y, axis_z)
{
    if (AccelerometerData.lastId === 0)
    { ... }
    else // Send GET using last measured Id
    {   ... }
}
```

`getLatestAccelerometerData` will be called once per second. The first time this function is called, we enter the `if` condition because we initialized `AccelerometerData.lastId` to `0`. In this condition, we want to get the newest accelerometer data. All subsequent calls to this function will enter the `else` condition because we will be setting `AccelerometerData.lastId` to the newest accelerometer data ID.

10 Smoothie Charts Builder: http://smoothiecharts.org/builder/



turnoff.us | Daniel Stori
Shared with permission from the artist

Listing 16.

```
1. function getLatestAccelerometerData (axis_x, axis_y, axis_z) {
2.   if (AccelerometerData.lastId === 0) {
3.     $.getJSON(Server.url,
4.       {},
5.       function (data, status) {
6.         if (status === 'success') {
7.           AccelerometerData.lastId = data.accelerationData.lastMeasurementId;
8.
9.           // In order to be compatible with the Safari browser, we cannot just pass in
10.          // the datetime string into the Date() constructor. Instead, we much pass in
11.          // each datetime component as a parameter to the Date() constructor
12.          let dateComps = data.accelerationData.accelerationMeasurements[0].dateTime.split(/[^0-9]/);
13.          let dateTime = new Date(dateComps[0], dateComps[1] - 1, dateComps[2],
14.            dateComps[3], dateComps[4], dateComps[5], dateComps[6]).getTime();
15.
16.          let xValue = data.accelerationData.accelerationMeasurements[0].axis.X;
17.          let yValue = data.accelerationData.accelerationMeasurements[0].axis.Y;
18.          let zValue = data.accelerationData.accelerationMeasurements[0].axis.Z;
19.
20.          axis_x.append(dateTime, xValue);
21.          axis_y.append(dateTime, yValue);
22.          axis_z.append(dateTime, zValue);
23.        }
24.      });
25.  } else // Send GET using last measured Id
26.  { ... }
27. }
```

Let's look at the `if` condition logic shown in Listing 16.

First, we request the latest accelerometer data from our web service endpoint by calling `jQuery.getJSON()` to get JSON data using an AJAX HTTP GET request. This request to `http://raspberrypi.local/accelerometerservice/` has no query parameters. Upon success, the *JSON* payload will be contained in the `data` parameter of the anonymous function. At this point we'll assign the `AccelerometerData.lastId` property to the value of the last measure ID:

```
AccelerometerData.lastId =
  data.accelerationData.lastMeasurementId;
```

Since we are only reading one set of measurements all the data we need is contained in `data.accelerationData.accelerationMeasurements[0]`.

The `TimeSeries` objects passed in require a `Date` object specifying the datetime of the data we want to append to the series. The `Date` object can be instantiated with the datetime string, however, in order to be compatible with multiple browsers, it is better to instantiate a `Date` object with the datetime string split into individual components as parameters to the `Date` constructor:

```
// In order to be compatible with the Safari browser, we cannot just pass in
// the datetime string into the Date() constructor. Instead, we must pass in
// each datetime component as a parameter to the Date() constructor
let dateComps = data.accelerationData.accelerationMeasurements[0].dateTime.split(/[^0-9]/);
let dateTime = new Date(dateComps[0], dateComps[1] - 1, dateComps[2],
       dateComps[3], dateComps[4], dateComps[5], dateComps[6]).getTime();
```

Next we will save the measurement data to local variables:

```
let xValue =
data.accelerationData.accelerationMeasurements[0].axis.X;
let yValue =
data.accelerationData.accelerationMeasurements[0].axis.Y;
let zValue =
data.accelerationData.accelerationMeasurements[0].axis.Z;
```

Finally we will append the `dateTime` `Date` object and the measurements to each `TimeSeries` object:

```
axis_x.append(dateTime, xValue);
axis_y.append(dateTime, yValue);
axis_z.append(dateTime, zValue);
```

Listing 17.

```
function getLatestAccelerometerData (axis_x, axis_y, axis_z) {
  if (AccelerometerData.lastId === 0)
  { ... }
  else // Send GET using last measured Id
  {
    $.getJSON(Server.url,
      {lastMeasurementId: AccelerometerData.lastId},
      function (data, status) {
        if (status === 'success') {
          AccelerometerData.lastId = data.accelerationData.lastMeasurementId;

          data.accelerationData.accelerationMeasurements.forEach(function (accelerationMeasurement) {
            // In order to be compatible with the Safari browser, we cannot just pass in
            // the datetime string into the Date() constructor. Instead, we much pass in
            // each datetime component as a parameter to the Date() constructor
            let dateComps = accelerationMeasurement.dateTime.split(/[^0-9]/);
            let dateTime = new Date(dateComps[0], dateComps[1] - 1, dateComps[2],
              dateComps[3], dateComps[4], dateComps[5], dateComps[6]).getTime();

            let xValue = accelerationMeasurement.axis.X;
            let yValue = accelerationMeasurement.axis.Y;
            let zValue = accelerationMeasurement.axis.Z;

            axis_x.append(dateTime, xValue);
            axis_y.append(dateTime, yValue);
            axis_z.append(dateTime, zValue);
          });
        }
      });
  }
}
```

Now in Listing 17 we will look at the `else` condition logic.

First, we request all the newest accelerometer data since the last measurement's ID from our web service endpoint by calling `jQuery.getJSON()` to get JSON data using an AJAX HTTP GET request and including the `AccelerometerData.lastId` property set as the query parameter `lastMeasurementId`:

```
http://raspberrypi.local/accelerometerservice/?lastMeasurementId=41877
```

Upon success, the *JSON* payload will be contained in the `data` parameter of the anonymous function. At this point, we'll assign the `AccelerometerData.lastId` property to the value of the`lastMeasurementId`:

```
AccelerometerData.lastId = data.accelerationData.lastMeasurementId;
```

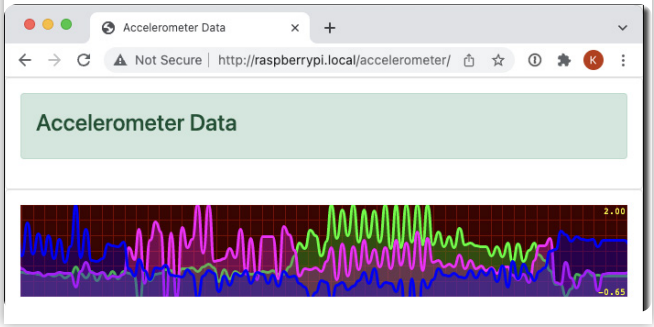
This block of code is essentially the same as that contained in the `if` condition, however we are reading (approximately) 10 measurements instead of just one. So we will iterate over `data.accelerationData.accelerationMeasurements` using the `forEach()` method:

```
data.accelerationData.accelerationMeasurements.forEach(function(accelerationMeasurement)
{
        ...
});
```

Listing 18.

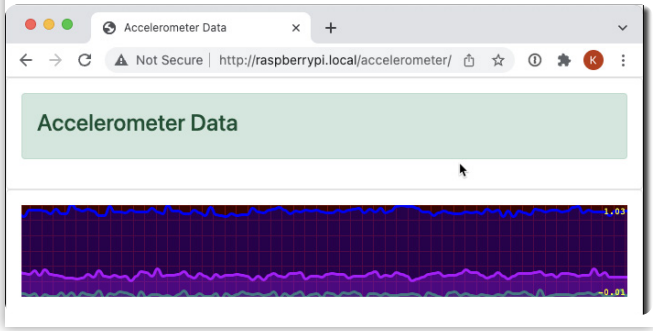
```
data.accelerationData.accelerationMeasurements.forEach(function(accelerationMeasurement)
{
    // In order to be compatible with the Safari
    // browser, we cannot just pass in the datetime
    // string into the Date() constructor.
    // Instead, we much pass in each datetime
    // component as a parameter to the Date()
    // constructor
    let dateComps =
        accelerationMeasurement.dateTime.split(/[^0-9]/);
    let dateTime =
        new Date(
                dateComps[0],
                dateComps[1] - 1,
                dateComps[2],
                dateComps[3],
                dateComps[4],
                dateComps[5],
                dateComps[6]
        ).getTime();

    let xValue = accelerationMeasurement.axis.X;
    let yValue = accelerationMeasurement.axis.Y;
    let zValue = accelerationMeasurement.axis.Z;

    axis_x.append(dateTime, xValue);
    axis_y.append(dateTime, yValue);
    axis_z.append(dateTime, zValue);
});
```

Figure 5. Accelerometer Data Plot



The rest of the code block (Listing 18) is essentially the same as the `if` block, except we are running it for each measurement (set as `accelerationMeasurement`) in the `data.accelerationData.accelerationMeasurements` array/

Figure 6. Data Plot of Dryer Running



Testing Our Plotting Application

Bring up a browser tab on your local computer and navigate to `http://raspberrypi.local/accelerometer/`. Gently shake your accelerometer sensor and change its orientation—you should see a real-time plot of all three axes of your accelerometer data displayed on the web page similar to Figure 5.

Conclusion

You now have a web application on your Raspberry Pi that displays a real-time plot of the activity your accelerometer is reading. If you have a clothes dryer, go ahead and put your Pi on top of it the next time dry a load of clothes, and take a look at the activity! See Figure 6.

In the next installment, we will:

- Install a State Machine that detects when our accelerometer stops shaking that we can use to detect when our clothes dryer is done
- and send a text message to ourselves using `sendmail` when our drying cycle is complete.

Ken Marks has been working in his dream job as a Programming Instructor at Madison College in Madison, Wisconsin, teaching PHP web development using MySQL since 2012. Prior to teaching, Ken worked as a software engineer for more than 20 years, mainly developing medical device software. Ken is actively involved in the PHP community, speaking and teaching at conferences. @FlibertiGiblets

Which License to Choose?

Chris Tankersley

Licensing for software, whether it is open source or not, is an integral part of releasing software. The commercialization of software has made it necessary for developers to be explicit in how users or other developers consume their software. Unfortunately, the topic of licensing is not as straightforward as many developers would like it to be.

As a quick refresher, licenses are the legal terms that an end-user must abide by to use the software legally. This includes installing, using, or integrating the software in other pieces of software. Common things the license covers are personal versus commercial use, how and where software may be installed, and whether or not the end-user may modify the software.

There is much more to licensing than just "is this software open source or not?" If you are releasing software, you can choose from hundreds of available licenses. Why are there so many, and what are the differences between them? What you choose affects how users can interact with your software.

As developers, we need to be keenly aware of the licenses that we use in our own software. Licenses differ in the liberties granted to a developer. And different licenses can have very serious legal repercussions for a company. Have you ever worked for a company with a strict "No GPL[1] Software" policy? There is a very real reason for that.

While software can be used anywhere in the world, my experience and view for this article will be United States-focused. I am not a lawyer. If you are unsure how something may work in your country, I would consult with a lawyer familiar with your local copyright laws. Consider the following descriptions a view on the intentions of the various licenses rather than hard legal advice.

No License

Have you ever come across a repository that you would love to use, but there is no license file? Congratulations, you have come across a library or project where you legally have no usage rights. You should avoid this project at all costs.

A core component of any license is the rights and rules around using a particular piece of software. If there is no license, the copyright holder holds all the cards. They have not granted you any usage rights, and they have not granted you access to the source code, even if you are looking at the code on GitHub or another service. They may come after you legally for using their intellectual property without permission.

While I would love to be altruistic about this, this is the world we live in, thanks to modern copyright laws. No license means no rights, even basic usage rights. The project is just taking advantage of lax policy policing by code repository providers.

Public Domain

The Public Domain[2] is not so much a license in-and-of-itself, but rather a declaration of the abandonment of copyright on a work. In the United States, this declaration is known as "dedicating." A work dedicated to the Public Domain is usually identified with some accompanying text of "This work is dedicated to the public domain." Once a work is dedicated to the public domain, the work is no longer owned by any entity and free for anyone to use.

The Public Domain is incredibly problematic from a legal standpoint. The first problem is that, like licensing, dedication to the Public Domain must be declared. Many countries, including the United States, assign copyright automatically to the author. There is no legal authority one is required to go through to establish copyright (though there are legal steps one can do to help protect and declare the copyright).

Nothing is automatically entered into the Public Domain except under a few conditions. A work enters the Public Domains by being dedicated by the original author as mentioned above, or if copyright expires. Copyright can expire either naturally or if the author does not file for an extension. Once copyright is removed, a work enters the Public Domain.

While this sounds straightforward, copyright is a complicated thing. The rules differ for people versus corporate entities. Copyright is transferrable, and if this is not meticulously documented, copyright ownership can get cloudy. Authors of works may or may not have done so "for hire" (how most software is developed). Corporate acquisitions and breakups can make it hard to know who owns what copyright. Shifting rules of the length of copyright can make it hard to know if a work is due to automatically age out to Public Domain status.

Assuming copyright ownership is actually known, there is no legal definition of what "Public Domain" is. There are rules for copyright as part of the Berne Convention of 1988, but countries can still enact their own rules. For example, The US does not require dedication for works before 1988 and

1 *GPL: https://www.gnu.org/licenses/gpl-3.0.en.html*

2 *The Public Domain: https://fairuse.stanford.edu/overview/public-domain/welcome*

declared anything before 1927 as Public Domain (with exceptions). Copyright is also handled country-by-country so that a work could be Public Domain in one country but not another. This presents a second hurdle for anyone wanting to use Public Domain software.

Even scarier? The copyright holder could potentially rescind Public Domain status[3] with the way US copyright works. Now, all of a sudden, a library your company uses is no longer Public Domain. What do you do? Since there was no license agreement, it is unknown, both theoretically and legally, what would happen if such a thing were to occur.

At the end of the day, much like with no license, Public Domain licensed software is a minefield and should be avoided.

Public Domain-like License

What happens when you want the legal requirements of a license but the freedom of Public Domain? You get a variety of Public Domain-like licenses that effectively say, "I do not care what you do with this software, and I am putting that in writing as the copyright holder." There are a handful of licenses available that fall under this category, but not all of these licenses are considered "Open Source."

The Creative Commons is probably the most popular suite of licenses from this type of software license. The Creative Commons is designed as a variety of licenses for authors to use to better control how their works are used. However, it is not recommended they be used for software[4]. Creative Commons 0[5] is the closest to a Public Domain declaration you can get while still providing legal text.

Another popular license is the Unlicense[6], which includes anti-copyright language to make it more applicable around the world. It includes an official copyright waiver as well as a "no warranty" statement, which is an important declaration missing from many of the Public Domain-like licenses. In fact, the Unlicense is considered open-source compatible, unlike other licenses in this category.

While there are a handful of Public Domain-like licenses, many are legally dubious. One of the more famous examples is the "Don't be a Dick[7]" license. On the surface, it looks like something akin to the Unlicense, but the main focus of the license is you are granted rights if you "aren't a dick." Unfortunately, there is no legal definition for what this means, and the license even mentions that the few examples given are not exhaustive. At any point, the copyright holder can just decide a user has broken the license based on whatever behavior the author deems is "dickish."

It is my opinion that short of a Public Domain-like license approved by the Open Source Initiative, you should avoid these types of licenses.

Proprietary License

A proprietary license is essentially going to be any license that is unique to an individual piece of software. For example, when you install a piece of software like Microsoft Office, you agree to what they call an "End User License Agreement." This agreement contains information that you might expect—it details how you may install the software, how usage is granted, and all kinds of other legal stuff.

While a company may copy-and-paste much of the text between their EULAs, each software's license governs just that piece of software. The Windows EULA does not cover Microsoft Office, and the Windows 10 EULA does not cover Windows 11 or previous versions of the software. The license is unique to that specific piece of software.

> *Are EULAs and Software Licenses different? It depends on who you ask, as they can cover many of the same topics. For what topics we are covering, it is pretty safe to equate an End User License Agreement to various other Software Licenses. One main difference is EULAs tend to not differentiate between source code and compiled code, where open-source licenses may be explicit on those two topics.*

Source Sharing

While many proprietary licenses restrict gaining access to the source code of a particular piece of software, that is not a defining factor of a proprietary license. Some software may come with an option called Source Sharing, where a software provider allows a customer access to the source code.

I have come across this mostly in enterprise software where the software provider sells software that solves many common problems for their customers, but customers may have very unique workflows or requirements. I used to work in the insurance industry, and all of our back-office software came with a source-sharing agreement. Each release of the software included a full copy of the source code we would patch with our custom changes and compile on our systems.

We would modify the software to work exactly how we needed it and work with the vendor and other customers to get our patches merged into the mainline code by the vendor. Customers were allowed to share their patches with the system among themselves. It was very much like a limited open-source ecosystem.

Where source sharing differs from Open Source licensing, the original vendor still controls the source. While we were granted access to the software, we could only share it with other customers. We were still required to pay a hefty licensing fee to get access to the software. Most damning of all, we found out after doing a license audit that the changes we made had an automatic copyright transfer to the vendor.

3 *rescind Public Domain status:* https://www.techdirt.com/?p=83508

4 *software:* https://phpa.me/creativecommons-can-i

5 *Creative Commons 0:* https://creativecommons.org/?p=12354

6 *Unlicense:* https://en.wikipedia.org/wiki/Unlicense

7 *Don't be a Dick:* https://dbad-license.org

What that meant was that the original vendor could, at any time, take our patches and sell them as their own. They could even take our patches and lock them behind an even more expensive license. While we had access to the source code, the proprietary license dictated that we did not own any of the code, even the code we wrote.

Open Source Licensing

Many people do not realize that despite open-source software arguably being the original way software was distributed, the term "Open Source Software" was not codified until 1998. This official definition is called "The Open Source Definition[8]" and was published by the Open Source Initiative as a copy of the Debian Free Software Guidelines[9]. These rules specify what makes a piece of software Open Source.

A collective known as the Open Source Initiative[10] is a group that helps govern and guide open-source software. This includes maintaining the "Open Source Definition" as well as providing various resources for open source projects. One of their most important projects is a list of open-source licenses that they consider compatible with the idea of Open Source.

For a piece of software to be considered open source, it must meet the following guidelines[11] from Wikipedia:

1. **Free redistribution**: The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. **Source code**: The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
3. **Derived works**: The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of the author's source code**: The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5. **No discrimination against persons or groups**: The license must not discriminate against any person or group of persons.
6. **No discrimination against fields of endeavor**: The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7. **Distribution of license**: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License must not be specific to a product**: The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
9. **License must not restrict other software**: The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
10. **License must be technology-neutral**: No provision of the license may be predicated on any individual technology or style of interface.

Smelly Open Source

Much like Public Domain-like licensing, many licenses look like open-source licenses but actually restrict what the user can do. The JSON License[12] is a perfect example with its famous line, "The Software shall be used for Good, not Evil."

What is legally Good, and what is legally Evil? Even outside of the legal definitions, an open-source license should not, and cannot, restrict the user in such arbitrary ways. A truly open source software license does not restrict what the user can do nor force the user to do specific things. Rule #6 of the Open Source Definition expressly invalidates anything licensed under licenses like the JSON license.

8 *The Open Source Definition:* https://en.wikipedia.org/wiki/The_Open_Source_Definition

9 *Debian Free Software Guidelines:* https://w.wiki/4z9Q

10 *Open Source Initiative:* https://opensource.org

11 *guidelines:* https://w.wiki/4z9P

12 *The JSON License:* https://www.json.org/license.html

Permissive Licenses

Permissive licenses are licenses that are not copyleft licenses (more about them in a moment) and allow proprietary derivative works. In many ways, this mirrors how software was originally handled—a developer puts software out into the world and allows anyone else to use it, even if that code gets locked up in proprietary software. The general idea is that the software being available makes the developer's life easier.

A few of the most common permissive licenses are the MIT[13], BSD[14], and Apache 2.0 licenses[15]. The MIT and BSD licenses closely resemble each other, though there are a variety of BSD licenses that have come out through the years. This general format is what has inspired many of the "smelly" open source licenses and shorter licenses.

You may see the BSD license also called the Original BSD License, and anywhere from the 0 Clause BSD License to a 4 Clause BSD license. Over the years, various rules surrounding licensed software have changed. For example, the 2 Clause BSD license drops a non-endorsement requirement that the 3 Clause BSD license includes. The 3 Clause BSD dropped an advertising requirement that the 4 Clause BSD license imposed. These days the 2 or 3 Clause BSD license is typically used.

Personally, I see permissive licenses working best in library or component code or places where the code is clearly intended to be used by other code. For example, I release my dragonmantank/cron-expression[16] library under the MIT license because it is meant to be used with someone else's code. I am more interested in solving a problem for a developer rather than making sure that the developer releases any changes back into the wild.

The fact that permissive licenses do allow for proprietary usage is a major downside to this type of license. One of the most famous examples of this was that Windows 2000 contained BSD licensed code[17] as part of the networking tools and stack. People were shocked at this, but Microsoft was well within their rights to use the code as long as they followed the license. And they did. If you are OK with allowing your code to be used this way, permissive licenses are a good selection for your code.

Copyleft Licenses

This brings us to Copyleft licenses. Copyleft licenses differ from permissive licenses in that they require derivative software to be licensed under the license of the software that was being integrated. As mentioned above, it is perfectly acceptable for permissive-licensed code to be used in software that does not share that same license (ala BSD code being used in proprietary code). On the other hand, copyleft software makes it a requirement that the code stay under the same license. Copyleft licenses tend to cater toward software freedom more than developer freedom.

The GPL[18] is usually the go-to example of a copyleft license. The GPL itself was born out of the frustration that proprietary software was causing to developers and how software was increasingly stripping developers of the rights they used to have. As part of this, the requirement that GPL-derived software must also be GPL licensed was a conscious decision. This decision forced developers that altered the software to distribute those changes when someone asked. In fact, a derivative license called the Affero GPL[19] (AGPL) even goes so far as to say that anyone that simply accesses the code can ask for the source code changes.

I find that copyleft licenses, especially the GPL itself, work best for full applications. Since applications tend to solve much larger problems and generally involve a huge amount of developer hours, it makes much more sense to keep ill actors from taking the open-source software and just renaming it and slapping a proprietary label. Imagine if the Linux kernel was released under the MIT license. Well, we know what happens.

Two popular operating systems are based on BSD code:

1. The macOS base operating system called Darwin[20]
2. The Orbis OS[21] that powers the Playstation 4 and Playstation 5

While Darwin started out very open, Apple increasingly slowed down upstream patches to the OS. While Apple is very upfront about its use of BSD software, very few realize that the Playstation is running a BSD-powered operating system. Sure, both Sony and Apple are legally following what the BSD license tells them to, but there is a vast amount of work that neither company is required to release back to the ecosystem.

So What do You do?

When you go to release software, think about your goal for users. Ask yourself this question, "should the software be proprietary or open-source?" While I wholeheartedly support open source and believe it is the way software should be distributed, I use an iPad and iPhone, and I use Windows for a lot of video gaming. I do a lot of open source work, but the vast majority of my life has been spent making proprietary software.

If you release software as open-source, do it the proper way. Use a license approved by the Open Source Initiative, and make sure you follow the Open Source Definition for your

13 *MIT: https://opensource.org/licenses/MIT*

14 *BSD: https://opensource.org/licenses/BSD-3-Clause*

15 *Apache 2.0 licenses: https://opensource.org/licenses/Apache-2.0*

16 *dragonmantank/cron-expression: https://github.com/dragonmantank/cron-expression*

17 *BSD licensed code: https://phpa.me/everything2-bsd-windows*

18 *GPL: https://opensource.org/licenses/GPL-3.0*

19 *Affero GPL: https://opensource.org/licenses/AGPL-3.0*

20 *Darwin: https://w.wiki/4wKH*

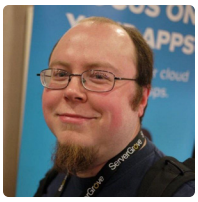
21 *Orbis OS: https://www.extremetech.com/?p=159476*

software. It is not much work, and you will find a license that fits your software's goals.

If you are just a developer, keep in mind what software you use and make sure you follow the license. Pay particular attention to what the dependencies of your dependencies require. Comcast has a license checker[22] that you can use to scan your `composer.lock` file to help suss out this information. Understand what it means to consume software under the different licenses.

I hope all this information helps clear up a lot of the misconceptions and unknown pitfalls of licensing. I would love to live in a world where we just share code and do not have to worry about the legalities of software design, but this is the world we live in. All I can say is help spread open-source software, and use it responsibly.

22 *license checker: https://github.com/Comcast/php-legal-licenses*



Chris Tankersley is a husband, father, author, speaker, podcast host, and PHP developer. Chris has worked with many different frameworks and languages throughout his twelve years of programming but spends most of his day working in PHP and Python. He is the author of Docker for Developers and works with companies and developers for integrating containers into their workflows. @dragonmantank

Related Reading

- *PHP is the Worst* by Chris Tankersley, September 2021
  https://phpa.me/tankersley-sept-2021
- *How to Build a REST API* by Chris Tankersley, May 2021
  https://phpa.me/tankersley-may-2021
- *Overriding Composer* by Chris Tankersley, October 2019
  http://phpa.me/education-oct-19

Operational Security

Eric Mann

It is remarkably easy to grow complacent in the digital world, but a lapse in security best practices inevitably leads to a lapse in security itself.

There was a lot of confusion in the early days of the cloud. Our websites started on either dedicated servers or poorly-compartmentalized shared hosts. Over time, virtual servers began to take over and better segmented unrelated workloads. Then tools like Amazon's AWS introduced the concept of a "managed cloud" of services that promised to make this kind of management easier.

For most workloads, it did. For those of us still learning how the cloud differed from a full server, it just made things confusing.

My first project was to migrate a particular website and API from a full server into the cloud. Rather than AWS, we selected an early version of Microsoft Azure. Doing so simplified our deployment as we didn't need to worry about licensing, .Net versions, or manually updating the GAC[1] to avail custom libraries to the platform.

This simplicity came at a cost.

Our application stack deployed in moments and was both highly available and remarkably stable. It was fast, efficient, and meant an end to 5 am wake-up calls because of literal fires in the facility hosting our old, physical server. The biggest downside—no email support.

On a dedicated server, I could trigger outgoing email messages directly. Early versions of Azure did not support SMTP directly, nor was managed transactional email a supported service at the time. My solution: open port 25 on a server in the rack down the hall and allow our Azure-hosted web service to relay messages through it.

It worked wonderfully!

Disaster strikes

The next morning, all of our support queue metrics were down. No one was getting messages. Customers were calling in, irate to have not received invoices, documentation, or even basic responses to questions. Digging in, I discovered that our corporate IP address—and thus our entire mail server—had been globally blacklisted.

I'd opened port 25 not just to Azure but to the entire world, and that world was gleefully using my open relay to spam the Internet.

As quickly as I'd opened things earlier, I locked that server down. Instead of routing messages through a server, I dumped them all to a file that I could pull manually. Then I spent the rest of the week pleading with blacklist maintainers to reinstate our IP's reputation so we could keep working.

In the end, Microsoft did release a managed transactional mail system, and I was able to fully integrate our application. This empowered our team to keep working while protecting us from accidentally exposing our internal mail system to would-be abusers elsewhere in the world.

I believed I could use our mail server in this way for a few days while I worked out an alternative solution. This mistaken belief was my undoing. A later review of system logs showed the abuse started literally *minutes* after I'd opened the server to incoming network connections.

My naive sense of "we'll be fine for a while" stood no chance against reality.

Learning from mistakes

Making mistakes is natural. It's how we learn, grow, and develop both personally and professionally. As disastrous as my email mistake turned out to be, it was one of the most formative lessons of my early career. I learned never to take *any* security stance for granted, lest my complacency opens the door to attackers.

Since then, I've seen countless developers fall prey to the same kind of naivety.

One developer was having trouble with the internal network mappings on a server for which they were responsible. Rather than ask for help, thus airing their inability to solve a problem, they elected to take the nuclear route. They opened their server to traffic from any IP address in hopes they could connect via SSH and solve the problem. The further mistake was opening *all ports* to all IP addresses.

An angry call from an ISP demanding to know why the server was participating in a DDoS attack later, and we had the server again locked down. We conducted a rather embarrassing post-mortem of the incident, and the developer, now understanding the ramifications of their actions, learned the lesson and moved on.

Another developer had trouble understanding how his network kept getting breached. His email gateway, messaging system, and firewall all showed multiple indications of compromise, but he couldn't figure out why. We'd completely wipe and reprovision systems, only to see alerts of malicious activity later that day. It wasn't until we got together on a screen share that the issue was apparent.

As he logged in, we all saw his password—`Passw0rd`.

1 *the GAC: https://phpa.me/microsoft-gac*

Another engineer pointed out how insecure that password really was. "But I have lower and upper case and a number. Wait, I need special characters, don't I?" He went to change his password, and sure enough, his current password was rated as "weak." He entered `Passw0rd!` instead and, greeted by the system's "strong" rating, assumed everything was golden.

This breach turned out to be a great opportunity to teach another engineer how to evaluate password strength[2], and he eventually changed to an *actual* strong password. Another round of system refreshes, and he hasn't seen any indication of compromise in ages.

Best practices and the ongoing quest for security

No developer can ever stop learning and say, "I know everything there is to know about software." Our industry is always growing, developing, and moving forward. To stop learning is to let the industry itself pass you by.

To be successful means exercising a practice of continual learning. You absolutely must learn new technologies, new languages, new frameworks. You must understand the best practices in your space today and keep your eyes open to catch their eventual evolution in the future. Adaptability is the name of the game and is a hard requirement for ongoing operations.

At least any operations that have a hope of any modicum of success.

Security in tech is no different from any other discipline. Things deemed secure today might be breached tomorrow as the industry learns, evolves, and adapts. There is never a state of "done" when it comes to security, so don't fool yourself into thinking security can be ignored. The tech world moves fast—you need to move faster to stay ahead.

Learn what it means to secure your network:

- Implement a strong firewall and don't open unnecessary ports to the world
- Focus on refreshing your understanding of entropy and password strength
- Use a password manager to eliminate human memory as the weak chain in your authentication scheme
- Implement multi-factor authentication[3].

Security is the responsibility of everyone involved in operations. As best practices continue to shift us towards a world where every developer is involved in operations, we're moving quickly into a space where we're *all* responsible for operational security as well.

Eric is a seasoned web developer experienced with multiple languages and platforms. He's been working with PHP for more than a decade and focuses his time on helping developers get started and learn new skills with their tech of choice. You can reach out to him directly via Twitter: @EricMann

2 *password strength:* https://phpa.me/password-strength

3 *authentication:* https://phpa.me/multi-factor-authentication

Accept Testing with Codeception

Joe Ferguson

Acceptance testing is my favorite tool to reach for when working with legacy applications that may have low test quality or no tests at all. Because acceptance testing approaches the application from outside of the source code, we're able to greatly increase test coverage without having to touch the application's code itself. Larger teams can use acceptance tests to prove new features behave as expected. For the single developer that knows they should be writing tests but still doesn't for whatever reason: acceptance testing can help them jump-start their application's test suite.

WHAT IS ACCEPTANCE TESTING

One of the most common test scenarios might be, "Can a user log in to our application and land on a specific URL we expect?" This scenario covers *many* methods in our application, giving us more test coverage at the expense of our test failures being more difficult to debug because so many methods are involved. If we find ourselves building a new feature, we can leverage acceptance testing to prove our feature works as expected, and we can also test our failure scenarios to prove our validation logic and form processing are what we expect.

Acceptance testing is a method of verifying our application behaves exactly as expected, often utilizing a web browser. We will write test scenarios that will be acted out by a user interacting with our application, such as logging in or performing a specific task. Acceptance tests are *slower* than unit tests because they rely on a web browser. What we lose in the speed of running acceptance tests we gain the confidence that our application works as designed with a real web browser. For small teams, acceptance testing can be your first step into a formal quality assurance (QA) process.

Whereas unit tests focus on testing isolated methods and feature tests focus on testing specific methods working together, acceptance testing exercises our application just as real users would by clicking on links, filling out forms, and processing data. We can create users easily, set permissions, and then verify those permissions are enforced. This level of detailed step-by-step testing provides incredible confidence the application behaves as expected. Acceptance testing is not a replacement for unit tests. Even in situations of legacy applications where acceptance tests are the only tests, we still write unit tests for new functionality.

How much of our tests should be unit or acceptance? Do we need 100% coverage? It depends. Test coverage is about confidence in your code. Does 100% unit test coverage give you confidence? What gives *me* confidence is knowing my applications behave as I expect them to. Acceptance testing also allows me to turn user stories into test scenarios and easily translate business requirements into acceptance tests. When our tests exercise the application in the exact same way real users will, the result is confidence in our test suite.

Acceptance testing isn't the solution for fast test feedback and should not replace unit tests. Acceptance tests should augment your existing test suites and add a layer of confidence that your users can interact with your application exactly as you've designed and tested.

Why Codeception[1] and not something like Behat[2] or Kahlan[3] Behat and other Behavior Driven Development (BDD) style testing frameworks are designed to test the application's behaviors. Read about Kahlan in the September 2018[4] issue. With Codeception's acceptance test suite, we can run scenarios with *real* users in *real* browsers. Codeception has been around the PHP ecosystem for 10 years or more, and the project has matured into a robust solution for full-stack testing. Codeception supports functional, unit, and API test suites out of the box, making it a perfect choice for new applications and legacy applications that may not have a test suite at all.

Getting Real

How we wire up this real web browser with Codeception is via Selenium[5]. Selenium is a suite of tools for automating web browsers, a very wide and open-ended topic. We will skip most of the configuration and complexity that often comes with Selenium by utilizing selenium-standalone[6] from NPM.

1 *Codeception:* https://codeception.com

2 *Behat:* https://docs.behat.org/en/latest/%3E

3 *Kahlan:* https://github.com/kahlan/kahlan

4 *September 2018:* https://www.phparch.com/magazine/2018/09/magniphpicent-7-3/

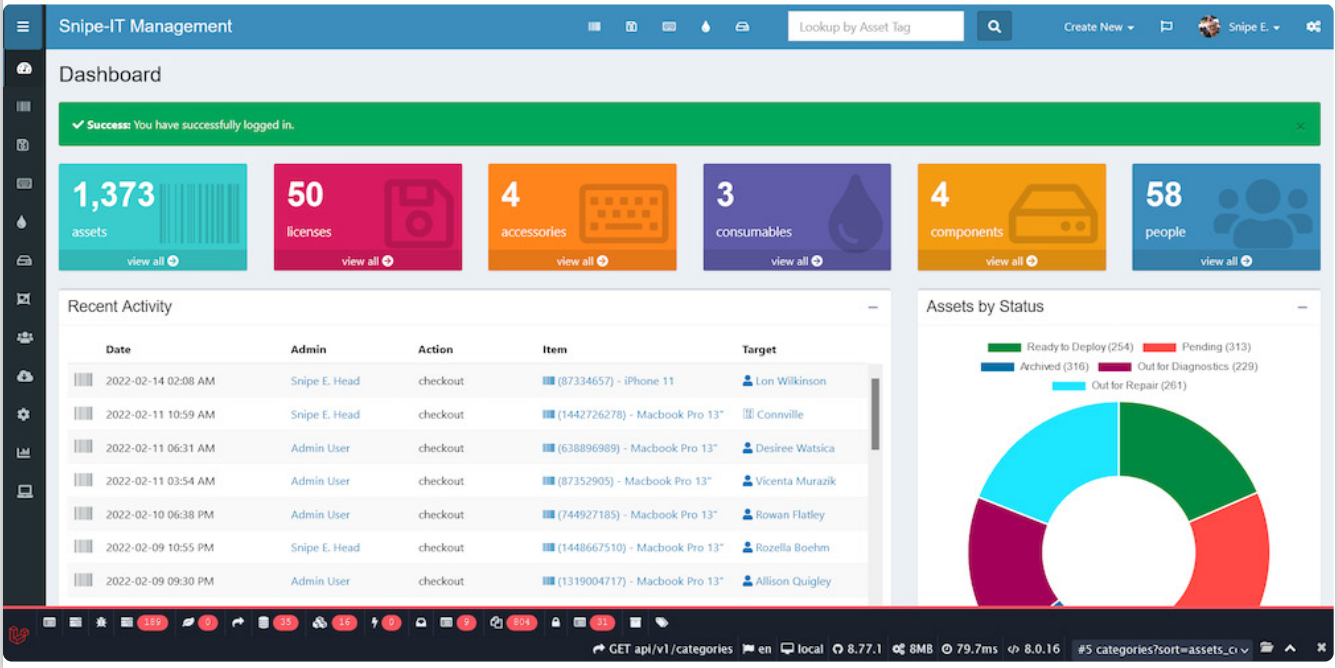
5 *Selenium:* https://www.selenium.dev/downloads/

6 *selenium-standalone:* https://www.npmjs.com/package/selenium-standalone

Figure 1.

```
selenium-standalone start
13:12:41.358 INFO [LoggingOptions.configureLogEncoding] - Using the system default encoding
13:12:41.361 INFO [OpenTelemetryTracer.createTracer] - Using OpenTelemetry for tracing
13:12:41.766 INFO [NodeOptions.getSessionFactories] - Detected 8 available processors
13:12:41.857 INFO [NodeOptions.discoverDrivers] - Discovered 3 driver(s)
13:12:41.871 INFO [NodeOptions.report] - Adding Firefox for {"browserName": "firefox"} 8 times
13:12:41.872 INFO [NodeOptions.report] - Adding Chrome for {"browserName": "chrome"} 8 times
13:12:41.872 INFO [NodeOptions.report] - Adding Edge for {"browserName": "MicrosoftEdge"} 8 times
13:12:41.893 INFO [Node.<init>] - Binding additional locator mechanisms: name, id, relative
13:12:41.907 INFO [LocalDistributor.add] - Added node 0d935aaa-d3dc-4959-8fc7-b8f981cc934b at http://172.22.24.67:4444. Health check every 120s
13:12:41.908 INFO [GridModel.setAvailability] - Switching node 0d935aaa-d3dc-4959-8fc7-b8f981cc934b (uri: http://172.22.24.67:4444) from DOWN to UP
13:12:42.032 INFO [Standalone.execute] - Started Selenium Standalone 4.0.0 (revision 3a21814679): http://172.22.24.67:4444
Selenium started
```

Figure 2.



There are multiple ways you can install selenium-standalone:

1. as a global NPM package—`npm install selenium-standalone -g`
2. save the package to your project—`npm install selenium-standalone --save-dev`
3. run via docker—`docker run -it -p 4444:4444 webdriverio/selenium-standalone`

If you have installed the NPM package the next step is to run `sudo selenium-standalone install` to download the Chromedriver and Selenium packages. By default, we'll get the Selenium, Chrome, Firefox, and Chromiumedge drivers installed for us to use. To start the service, we'll use `selenium-standalone start`, which will listen on port 4444 by default. See Figure 1.

Since we're using *real* browsers to test our application, I felt it was fitting to pull a *real* PHP project: Snipe-IT. Snipe-IT is an open-source IT asset/license management system as shown in Figure 2. Think of large companies with thousands of laptops, desktops, monitors, Windows licenses, and more deployed to their workforce. Snipe-IT is a tool they can use to help manage all of that headache. The project was so successful snipeyhead[7] founded a company[8] to provide a hosted version and offer support to customers. Over the years, I've been able to contribute to the project, and it's an incredible example of a modern Laravel based application. I'm currently using PHP 8.0 and working off the `develop` branch due to a new major version release (6.0!) coming in the near future. My development environment is Ubuntu 20.04 LTS; however, these commands should translate to macOS directly.

I have my Snipe-IT running at http://snipe-it.test[9], and once upon a time, the project featured Codeception. But if you're just getting started with Codeception, you can use `composer require "codeception/codeception" --dev` to install the package. Make sure you review the fantastic documentation[10] for all the configuration options.

7 *snipeyhead:* https://twitter.com/snipeyhead

8 *company:* https://snipeitapp.com/company

9 *http://snipe-it.test:* http://snipe-it.test

10 *documentation:* https://codeception.com/docs

Because Codeception has previously been used, we'll review the configuration from the root of the project and verify we're ready to run the Acceptance test suite found in the `tests/Acceptance` folder of the project. The root configuration defines where the Codeception test suites will be located along with support and data directories, and the extension configuration can be leveraged to run only failed tests. Listing 1 shows these configurations, which can be found in `codeception.yml`.

Listing 1.

```
paths:
    tests: tests
    output: tests/_output
    data: tests/_data
    support: tests/_support
    envs: tests/_envs
actor_suffix: Tester
extensions:
    enabled:
        - Codeception\Extension\RunFailed
```

Our acceptance test suite has its own configuration file to specify how this suite should operate. Most importantly, we configure the `WebDriver` module to use the URL for our local development environment and specify the `chrome` browser. We're also going to configure the Laravel module[11] to utilize the ORM, so we have access to Eloquent in order to perform any setup or teardown actions we may need for specific scenarios.

The configuration file containing Listing 2 is located at `tests/acceptance.suite.yml`. Codeception uses these Yaml configuration files to create generated code helpers for your test suite. When you make a change to a configuration file, you need to run `php vendor/bin/codecept build` (Figure 3) to build the support classes Codeception needs.

Figure 3.

```
┌─(~/Code/snipe-it)(acceptance-testing-codeception S:9 no-remote)──────
└─ ./vendor/bin/codecept build
Building Actor classes for suites: acceptance, api, functional, unit
 -> AcceptanceTesterActions.php generated successfully. 109 methods added
\AcceptanceTester includes modules: WebDriver, Laravel, \Helper\Acceptance
 -> ApiTesterActions.php generated successfully. 196 methods added
\ApiTester includes modules: \Helper\Api, REST, Laravel5, Asserts
 -> FunctionalTesterActions.php generated successfully. 153 methods added
\FunctionalTester includes modules: \Helper\Functional, Laravel5, REST
 -> UnitTesterActions.php generated successfully. 245 methods added
\UnitTester includes modules: \Helper\Unit, Asserts, Laravel5
```

The Snipe-IT project makes use of Laravel's database seeders so we can set up our application with realistic data generated randomly by running `php artisan db:seed`. The seeding will also create an administrative user with the username of `snipe` and password of `password`, which we can use to validate our user's ability to log in to our application. There's

11 Laravel module: https://github.com/Codeception/module-laravel5

Listing 2.

```
actor: AcceptanceTester
modules:
    enabled:
        - WebDriver:
            url: <http://snipe-it.test>
            browser: chrome
            capabilities:
                chromeOptions:
                    args: ["--headless", "--disable-gpu"]
        - Laravel:
            part: ORM
            cleanup: false # can't wrap into transaction
        - \\Helper\\Acceptance
```

an existing `tests/acceptance/LoginCest.php` test scenario we can start with, shown in Listing 3.

Listing 3.

```
<?php

class LoginCest
{
    public function _before(AcceptanceTester $I)
    {
    }

    // tests
    public function tryToLogin(AcceptanceTester $I)
    {
        $I->wantTo('sign in');
        $I->amOnPage('/login');
        $I->see(trans('auth/general.login_prompt'));
        $I->seeElement('input[type=text]');
        $I->seeElement('input[type=password]');
    }
}
```

The `_before` method can be used to do any setup required before we start executing our test cases. In our `LoginCest`, we open the login page, expect to see some text and then see form elements for password and text fields. We can give this test a run by itself via `php vendor/bin/codecept run tests/acceptance/LoginCest.php`

Assuming everything went well, Figure 4 shows the console output we should see.

Figure 4.

```
┌─(~/Code/snipe-it)(acceptance-testing-codeception S:9 no-remote)──────
└─(1)─ ./vendor/bin/codecept run tests/acceptance/LoginCest.php
Codeception PHP Testing Framework v4.1.29
Powered by PHPUnit 9.5.11 by Sebastian Bergmann and contributors.

Acceptance Tests (1) ------------------------------------------------
✓ LoginCest: Sign in (0.91s)
---------------------------------------------------------------------


Time: 00:03.730, Memory: 32.50 MB

OK (1 test, 3 assertions)
```

Figure 5.

```
┌─(~/Code/snipe-it)(acceptance-testing-codeception M:1 S:9 no-remote)─
└─ ./vendor/bin/codecept run --debug tests/acceptance/LoginCest.php
Codeception PHP Testing Framework v4.1.29
Powered by PHPUnit 9.5.11 by Sebastian Bergmann and contributors.

Acceptance Tests (1) ------------------------------------------------
Modules: WebDriver, Laravel, \Helper\Acceptance
---------------------------------------------------------------------
LoginCest: Sign in
Signature: LoginCest:tryToLogin
Test: tests/acceptance/LoginCest.php:tryToLogin
Scenario --
 I am on page "/login"
   [GET] http://snipe-it.test/login
 I see "Please Login"
 I see element "input[type=text]"
 I see element "input[type=password]"
 PASSED

---------------------------------------------------------------------


Time: 00:05.308, Memory: 32.50 MB

OK (1 test, 3 assertions)
```

We can use the debug flag to get more output from Codeception as shown in Figure 5.

Now that we can verify our login pages loads let's expand our LoginCest test by filling out the form and attempting to login to the application. We'll add steps to our `tryToLogin()` method to `fillField()` the username and password fields, `click()` the Login button, and `see()` the success string. See Listing 4.

Listing 4.

```
1.  public function tryToLogin(AcceptanceTester $I)
2.  {
3.      $I->wantTo('sign in');
4.      $I->amOnPage('/login');
5.      $I->see(trans('auth/general.login_prompt'));
6.      $I->seeElement('input[type=text]');
7.      $I->seeElement('input[type=password]');
8.      $I->fillField('username', 'snipe');
9.      $I->fillField('password', 'password');
10.     $I->click('Login');
11.     $I->see('Success: You have successfully logged in.');
12. }
```

Figure 6.

```
┌─(~/Code/snipe-it)(acceptance-testing-codeception M:2 S:9 no-remote)─
└─ ./vendor/bin/codecept run --debug tests/acceptance/LoginCest.php
Codeception PHP Testing Framework v4.1.29
Powered by PHPUnit 9.5.11 by Sebastian Bergmann and contributors.

Acceptance Tests (1) ------------------------------------------------
Modules: WebDriver, Laravel, \Helper\Acceptance
---------------------------------------------------------------------
LoginCest: Sign in
Signature: LoginCest:tryToLogin
Test: tests/acceptance/LoginCest.php:tryToLogin
Scenario --
 I am on page "/login"
   [GET] http://snipe-it.test/login
 I see "Please Login"
 I see element "input[type=text]"
 I see element "input[type=password]"
 I fill field "username","snipe"
 I fill field "password","password"
 I click "Login"
 I see "Success: You have successfully logged in."
 PASSED

---------------------------------------------------------------------


Time: 00:04.220, Memory: 32.50 MB

OK (1 test, 4 assertions)
```

Figure 7.

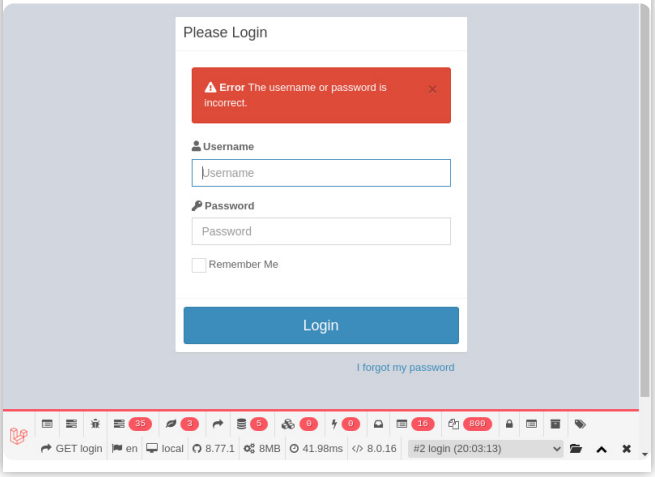
```
There was 1 failure:

---------
1) LoginCest: Sign in
Test  tests/acceptance/LoginCest.php:tryToLogin
Step  See "Success: You have successfully logged in."
Fail  Failed asserting that  on page /login
--> Please Login
×
Error The username or password is incorrect.
Username
Password
Remember Me
Login
I forgot my password
35
3
5
0
0
16
800
#1 login (stacked) (20:03:13)
#2 login (20:03:13)
8.0.16
41.98ms
8MB
8.77.1
local
en
GET login
--> contains "Success: You have successfully logged in.".

Scenario Steps:

 8. $I->see("Success: You have successfully logged in.") at tests/acceptance/LoginCest.php:20
 7. $I->click("Login") at tests/acceptance/LoginCest.php:19
 6. $I->fillField("password","password") at tests/acceptance/LoginCest.php:18
 5. $I->fillField("username","joe") at tests/acceptance/LoginCest.php:17
 4. $I->seeElement("input[type=password]") at tests/acceptance/LoginCest.php:16
 3. $I->seeElement("input[type=text]") at tests/acceptance/LoginCest.php:15

Artifacts:

Png: /home/halo/Code/snipe-it/tests/_output/LoginCest.tryToLogin.fail.png
Html: /home/halo/Code/snipe-it/tests/_output/LoginCest.tryToLogin.fail.html

FAILURES!
Tests: 1, Assertions: 4, Failures: 1.
```

Running our tests confirms our user can log in as shown in Figure 6.

What if we changed our username from `snipe` to `joe`? Let's run our tests again and see an example (Figure 7) of a test failure.

We can see from the output that our username or password is incorrect, as we expected. The added bonus that Codeception brings is that we have the HTML output from our application AND a screenshot of when our test failed. We can inspect the contents of `tests/_output` and see `LoginCest.tryToLogin.fail.html` and as shown in Figure 8 `LoginCest.tryToLogin.fail.png`.

Figure 8.



Armed with the full HTML output from our application and a screenshot of what the browser saw, we can debug what potentially went wrong. In our example test case, we can verify the user is experiencing the "username or password is incorrect" error.

You might notice a lot of output happening in the window where you ran `selenium-standalone start`. This window is the back end of Selenium running our headless browser to execute our tests. If you run into a test failure that doesn't make sense, we can review these logs to see if something in Selenium was breaking. In *most* cases, you should be able to safely ignore this output.

It's important to understand what we're testing. With our acceptance test `LoginCest.php`, we are testing a lot of different parts of the application as well as UI elements, log-in form validation, database access, and generally, everything required to allow a user to log in to our application is fully functional. This test is how we can gain large amounts of test coverage of our codebase at a tradeoff of the test failures being more obscure and possibly more difficult to debug because so many methods are involved. This type of test is how we can validate all of our authentications, and login-related unit tests work when we put them all together and authenticate and redirect a user. If you're building these tests in a legacy application that does not have an existing test suite, you gain confidence in your refactoring work by writing tests to validate the system is working in its current state. You *hope* the current state is working, but it may not always be; the *known* state is more important here.

Codeception also supports using helper methods such as logging in a user to make this code easily reusable. We can add helper methods to `tests/_support/AcceptanceTester.php`, and we see there is an existing `test_login()` method that attempts to login our `snipe` user. Our static `test_login` method contains:

```
public static function test_login($I)
{
    $I->amOnPage('/login');
    $I->fillField('username', 'snipe');
    $I->fillField('password', 'password');
    $I->click('Login');
}
```

With this helper method, we could refactor our `tryToLogin()` test to contain our helper method. We could also add our expanded test to the helper, so we know we're validating the login actually works for our test user.

```
public function tryToLogin(AcceptanceTester $I)
{
    $I->test_login($I);
}
```

Running our test again (Figure 9) validates our `LoginCept.php` test functions as expected.

Figure 9.

```
┌─(~/Code/snipe-it)(acceptance-testing-codeception M:2 S:9 no-remote)─
└─(1)─ ./vendor/bin/codecept clean
Cleaning up output /home/halo/Code/snipe-it/tests/_output/...
Done

┌─(~/Code/snipe-it)(acceptance-testing-codeception M:2 S:9 no-remote)─
└─ ./vendor/bin/codecept run tests/acceptance/LoginCest.php
Codeception PHP Testing Framework v4.1.29
Powered by PHPUnit 9.5.11 by Sebastian Bergmann and contributors.

Acceptance Tests (1) ------------------------------------------------
✓ LoginCest: Try to login (1.89s)
---------------------------------------------------------------------


Time: 00:04.250, Memory: 32.50 MB

OK (1 test, 0 assertions)
```

We can utilize these helper methods to log in administrators as well as less privileged users to verify our permissions are being enforced without having to copy and paste the same few lines of code in each of our test cases.

Everyone tests their application. Some of us automate it. You test your application when you deploy it and then click around production to "make sure nothing obvious is broken." What if we could know there's a problem *before* we deploy our code? Codeception gives you the ability to open your application in your browser of choice and click around and use the system exactly as your real-world users would. With selenium-standalone, you can skip most of the complexity typically involved in Selenium. If you don't already have a quality assurance team, acceptance tests can be your start. If you're already working with a QA team, Codeception is a fantastic way to expand your test coverage directly from the people already testing your application manually. If you're not testing your application in an automated way, start today with Codeception and start small. Test coverage doesn't happen instantly but you will gain confidence as your test suite expands to cover more of your application.

Michael Bodnarchuk[12] built Codeception in Kyiv, Ukraine. The PHP world is better because of Codeception. Make tests, not war.

Joe Ferguson is a PHP developer and community organizer. He is involved with many different technology related initiatives in Memphis including the Memphis PHP User group. He's been married to his extremely supportive and amazing wife for a really long time and she turned him into a crazy cat man. They live in the Memphis suburbs with their two cats. @JoePFerguson

12 *Michael Bodnarchuk: https://twitter.com/davert*

When the New Requirement Arrives

Edward Barnard

When the new requirement arrives, does this mean we cram it in the best place, making our software more difficult to work with, or might we have ways to make the result cleaner than before we started? We'll take advantage of Martin Fowler's concept of Preparatory Refactoring.

This article will be short and to the point—because we planned it that way—without knowing what the new requirement might be.

> *"I love it when a plan comes together." George Peppard as Colonel Hannibal Smith leading the A-Team (1983)*

Over the past three "DDD Alley" articles, we have been developing production code and tests for the athlete Season Registration workflow. The new requirement, which I'm about to describe, is perfectly reasonable and, in fact, we perhaps should have seen it coming.

New Requirement

Once an athlete completes registration, we send an email confirmation. That's perfectly normal and expected behavior for a website, right? What's more difficult here is determining who needs to receive that email. The athlete registration might or might not require the athlete's email address based on age and other considerations (under age 13, under age 18, etc.). That athlete might or might not have guardian information. This requirement, again, is based on athlete age and other considerations.

Therefore, one part of the "send email confirmation" requirement is determining recipients based on business rules. Those are the same business rules we discussed previously ("Turning to Domain-Driven Design" in January 2022 php[architect]), so we won't repeat the discussion here.

The other part of our new requirement is to retrieve the relevant email addresses from our database. This retrieval, at first glance, is easy. Why? Because those email addresses are in the same tables already being accessed via our "role-based lookups" class.

Solid Principles

Not so fast! We have, to this point, been structuring our architecture around the SOLID[1] principles of object-oriented programming. The "O," the Open-closed principle[2], explains that software should be open to extension but closed to modification. What does that mean to us here?

The Open-Closed Principle means that when we receive a new use case, we should aim to implement that new use case without touching any existing software. We should (in theory) be adding files without modifying existing files.

Our first instinct was to extend (i.e., edit) the role-based lookups class. Sure, it gets bigger and more bloated, but that gets our new feature out the door as quickly as possible. Can we do better? Yes, we can!

Bloated Class

We walked through a long discussion of this very possibility last month ("Better Late Than Never" in March 2022 php[architect]). We noted Martin Fowler's advice for our situation (*Refactoring: Improving the Design of Existing Code*, 2nd edition, p. 82):

> *The clients of such a class are often the best clue for splitting up the class. Look at whether clients use a subset of the features of the class. Each subset is a possible separate class. Once you've identified a useful subset, use "Extract Class", "Extract Superclass", or "Replace Type Code with Subclasses" to break it out.*

Our new requirement (which is to lookup email addresses) uses a subset of the features of the existing class. The fact, that we're using a subset of the features of the existing class, is our signal to consider some sort of refactoring.

Kent Beck explains[3] the approach we'll take:

> *For each desired change, make the change easy (warning: this may be hard), then make the easy change.*

Martin Fowler calls Beck's advice preparatory refactoring[4]:

> *This is where I'm adding a new feature, and I see that the existing code is not structured in such a way that makes adding the feature easy. So first I refactor the code into the structure that makes it easy to add the feature.*

As Beck warns, our change may be difficult. Why is this? Because we have full unit test coverage. Our unit tests are in lockstep with our production code. Our first task, then, is to create a proper test boundary.

1 *SOLID:* https://en.wiki/wiki/SOLID

2 *Open-closed principle:* https://w.wiki/3AG4

3 *explains:* https://phpa.me/twitter-kent-beck-change

4 *preparatory refactoring:* https://phpa.me/martinfowler-refactoring-ex

Test Boundary

Robert C. Martin explains in *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, pp. 251 and 252:

> *The issue, of course, is coupling. Tests that are strongly coupled to the system must change along with the system. Even the most trivial change to a system component can cause many coupled tests to break or require changes…*
>
> *The way to accomplish this goal is to create a specific API that the tests can use to verify all the business rules… The purpose of the testing API is to decouple the tests from the application… The goal is to decouple the* **structure** *of the tests from the* **structure** *of the application. [Emphasis in the original.]*

It's time to extract an interface.

But first, let's note the terminology I'm about to use here. We have a possibility of confusion by calling this refactoring "Extract Interface."

- Fowler's *Refactoring*, second edition, explains "Extract Class" and mentions adjusting interfaces (p. 183) as part of the "Extract Class" refactoring.
- However, Fowler's *Refactoring*, first edition, includes the "Extract Interface" refactoring (p. 341). That is not what we're doing here.
- PhpStorm, on its "Refactor" menu, includes "Extract/Introduce > Interface…". This PhpStorm operation is what we're doing here.

Why are we extracting an interface? The unit tests, and only the unit tests, will use the interface. Does it make sense to create a production interface that is only being used by the automated tests? Absolutely! Yes!

More importantly, extracting that interface allows the tests to evolve in one direction while the production code possibly evolves in a different direction. The interfaces describe the needs of the individual clients of our "role-based lookups" class. We are fulfilling the "I" in "SOLID," which is the interface segregation principle[5]:

> *No code should be forced to depend on methods that it does not use.*

Preparatory Refactoring

Here are the steps of our preparatory refactoring:

- Run all the unit tests. They should all be passing before we begin to refactor.
- Create an interface based on the role-based lookup class's public methods. I used PhpStorm's built-in "Refactor > Extract/Introduce > Interface" tool to do so.
- Change the role-based lookup class to implement the interface.
- Run the unit tests. They should all pass as-is.
- Change the unit tests to reference the interface.
- All tests should still pass.

5 *interface segregation principle:* https://en.wikipedia.org/wiki/Interface_segregation_principle

Implement Feature

Now we can easily support our new feature. Note that the following steps do not change existing code. Also, note that the above steps (the preparatory refactoring) did not change any existing functionality.

- Define the public methods we need for the new feature, i.e., the methods extracting email addresses from the database.
- Create an interface describing those methods.
- Create an empty class implementing that interface and extending the original role-based lookups class.
- All current tests should still pass.
- Develop tests and production code implementing that new feature.

Listing 1 shows the new interface.

The beginning of the implementation looks like Listing 2.

Listing 1.

```
<?php

declare(strict_types=1);

namespace App\BoundedContexts\AMS\DomainModel\Interfaces\RoleBasedLookupInterfaces;

interface ILookupRecipientEmail
{
    /** covered */
    public function alternateGuardianEmail(): string;

    /** covered */
    public function alternateGuardianExists(): bool;

    /** covered */
    public function isEmailAddressRequired(): bool;

    /** covered */
    public function leagueRequiresGuardian(): bool;

    /** covered */
    public function primaryGuardianEmail(): string;

    /** covered */
    public function primaryGuardianExists(): bool;

    /** covered */
    public function userEmail(): string;

    /** covered */
    public function userIsUnder18(): bool;
}
```

Extract Superclass

With all unit tests in place and passing, I realized that I could take one more step to make the code easier to understand and follow. I moved the "memoize" methods into a parent class.

Note that we created the new feature first. Only after the new feature was working, with full test coverage in place, did we move the "memoize" support to a parent class.

Refactoring, 2nd edition page 376, lists the steps to take for the "Extract Superclass" refactoring.

After creating that superclass, the main role-based lookups class now starts like Listing 3.

Listing 2.

```
class RecipientEmail extends RoleBasedLookup
            implements ILookupRecipientEmail
{
    public function alternateGuardianEmail(): string
    {
        $this->loadParticipant();
        if (null === $this->participant) {
            return '';
        }
        $this->loadAlternateGuardianEmailAddress();
        if (null === $this->alternateGuardianEmailAddress) {
            return '';
        }
        return $this->alternateGuardianEmailAddress->email;
    }
}
```

Listing 3.

```
/**
 * This is a singleton for each UserRole so as to
 * cache objects as we fetch them
 */
class RoleBasedLookup extends RoleBasedMemoize
            implements CRole, CProfile, IRoleBasedLookup
{
    /** covered */
    public function userFirstName(): string
    {
        $this->loadUser();
        if (null === $this->user) {
            return '';
        }
        return $this->user->first_name;
    }
}
```

Listing 4.

```
1.    /** covered */
2.    public static function getInstance(UserRole $userRole): static
3.    {
4.        $uniques = [
5.            $userRole->user_id,
6.            $userRole->role_id,
7.            $userRole->team_id,
8.            static::class,
9.        ];
10.       $unique = implode(':', $uniques);
11.       if (!array_key_exists($unique, self::$instances)) {
12.           self::$instances[$unique] = new static($userRole);
13.       }
14.       return self::$instances[$unique];
15.   }
```

Can we pull the `getInstance()` method up to the superclass? Yes we can, using PHP's late static binding. The base class `RoleBasedMemoize` contains the code shown is Listing 4.

In other words, `RecipientEmail::getInstance()`, `RoleBasedLookup::getInstance()`, and any test fixtures will produce the correct class instance.

Why was adding a new feature without bloating the existing class so easy? Because we had full test coverage in place. We, therefore, took each step with confidence that nothing broke.

Summary

I enjoy learning from other peoples' experiences. It's often less painful than learning directly! We took outside advice on:

- The Open/Closed principle - it's better to add new code and leave existing code untouched
- The Interface Segregation principle - it's better to have small interfaces focused on a specific need
- When and why to break up a large class
- Making the change easy (which may be hard), then making the easy change; Fowler calls this preparatory refactoring
- Extracting interfaces when needed as the testing boundary

With the preparatory refactoring complete, we then created an entirely new class to contain our new feature. That was easy.

Why was it easy? Because we planned it that way. We had full test coverage in place to protect us during future refactoring.

I love it when a plan comes together.

Ed Barnard had a front-row seat when the Morris Worm took down the Internet, November 1988. He was teaching CRAY-1 supercomputer operating system internals to analysts as they were being directly hit by the Worm. It was a busy week! Ed continues to indulge his interests in computer security and teaching software concepts to others.
@ewbarnard

Drupal 9 - Introduction and Installation

Nicola Pignatelli

Hello everyone. Welcome to the first of a series of articles that will take you into the world of Drupal, one of the most popular Open Source CMS used for various installations types, from the personal blog to the corporate site up to ad hoc software to solve specific requests. Versions currently installed on the web are 7, 8, and 9.

From the list below, you can see the EOL of any version. It must be remembered that version 7 was supposed to reach the EOL in November 2021. But due to COVID-19, it was extended to November 2023 to accommodate the companies and associations that use it and could not afford the update with a monetary invention.

Version	End of Life
7.0	November 2023
8.0.0, 8.1.0, 8.2.0, 8.3.0	November 2021
9.0	EOL not defined
10.0	not defined

Difference Between Drupal Versions

Drupal 8 introduced the Symfony framework within the core, the Twig framework for frontend templating, page loading speed is improved thanks to entity caching, and javascript is now loaded only when necessary.

Drupal[1] 9 is built on top of version 8, allowing for an easier upgrade.

Drupal 10, expected to be available in June 2022, has two key improvements. The use of Symfony 5 or 6 replacing version 4, and the mandatory use of PHP 8 to keep the system secure.

Installation

In the article, I will explain how to install Drupal 9. I will hypothesize using a pc/server with Linux Ubuntu installed. I would strongly discourage using a Windows system, but if you do choose a Windows installation, check out Install Drupal on Windows[2] for some guidance.

Requirements for Drupal 9

- PHP 7.3+. PHP[3] 8 is supported from Drupal 9.1.0
- Web Server
  - Apache[4] 2.4.7+
  - Nginx[5] 0.7.x+
- Database Server
  - MySQL[6] or Percona 5.7.8+
  - MariaDB[7] 10.3.7+
  - SQLite 3.26+ is required
  - PostgreSQL 10+ with the pg_trgm extension
- Drush[8]
  - version 10+ requires PHP 7.1+
  - version 11+ requires PHP 7.4+

Other Software

- Composer[9]

Download Drupal 9 Via Composer

Composer is a tool to install/manage the libraries on which your project depends. Remember that installing Drupal 8.0 or later with tarball is not a problem because starting with Drupal 8.0, Composer was pre-installed. Run the commands in Listing 1 to download and install Composer.

Listing 1.

```
1. php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
2. php -r "if (hash_file('sha384', 'composer-setup.php') ===
'906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac310e8cba5b96ac8b64115ac402c8cd292b8a03482574915d1a8')
{ echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); }
echo PHP_EOL;"
3. php composer-setup.php
```

1 *Drupal:* http://www.drupal.org

2 *Install Drupal on Windows:* https://www.drupal.org/node/2347717

3 *PHP:* http://www.php.net

4 *Apache:* http://www.apache.org

5 *Nginx:* http://www.nginx.org

6 *MySQL:* https://dev.mysql.com/downloads/mysql/

7 *MariaDB:* https://mariadb.org

8 *Drush:* https://www.drush.org/latest/install

9 *Composer:* https://getcomposer.org

The last line will generate the composer-phar file. This file is the Composer binary. It is a PHAR (PHP archive), an archive format for PHP that can be run on the command line. If you want to install the composer-phar file in a particular directory, launch the setup with the following command:

```
php composer-setup.php --install-dir=bin \
--filename=composer
```

Otherwise let's copy it manually:

```
sudo mv composer.phar /usr/local/bin/composer
```

To clear the installation:

```
php -r "unlink('composer-setup.php');"
```

To Install Drupal, launch the following command:

```
composer create-project \
drupal/recommended-project site.dev
```

The previous command will download the current official release of Drupal. If you want a different version, add the version number to the command after a colon. For example, to download version 9.3.6:

```
composer create-project \
drupal/recommended-project:9.3.6 site.dev
```

A list of all releases is available from Drupal. For the latest 9.3 version, use `drupal/recommended-project:^9.3` in the above command, or `drupal/recommended-project:^9` for the latest Drupal 9 version.

Install Drupal Using the Standard Web Interface

After Composer finishes downloading the packages, you can navigate your browser to your site's URL and start the setup. It'll ask for the database credentials, a name for the admin user, and some basic information.

Configure Apache

Install Apache using the following command.

```
sudo apt install apache2
```

Write into file `/etc/hosts` this line:

```
127.0.0.1 site.dev
```

You can configure Apache to use a virtual host. In Listing 2, I show a directive to use in file `/etc/apache2/sites-available/site.dev.conf`.

Use the following code to create a symbolic link to this file and reload apache configuration:

```
sudo ln -s /etc/apache2/sites-available/site.dev.conf \
/etc/apache2/sites-enabled/site.dev.conf

sudo service apache2 reload
```

Configure Mariadb

The following commands install a database.

```
sudo apt install mariadb-server
sudo apt install mariadb
```

To secure MariaDB, run the following command:

```
sudo mysql_secure_installation
```

Login to MariaDB:

```
sudo mysql -uroot -p
```

Execute the following commands to create a database and grant privileges.

```
CREATE DATABASE db_test;
GRANT ALL PRIVILEGES ON db_test.*
    TO 'test'@'localhost'
    IDENTIFIED BY 'test1234';
FLUSH PRIVILEGES;
```

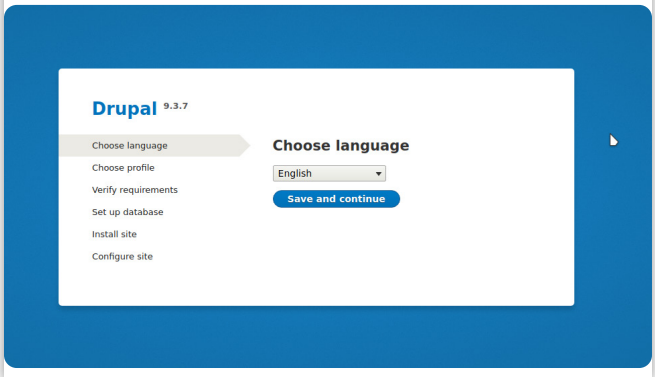
Install Drupal 9 Via a Web Interface

You should see Figure 1 when you open your browser and go to http://site.dev/install.php.

Listing 2.

```
1. <VirtualHost *:80>
2.     ServerAdmin admin@site.dev
3.     DocumentRoot /var/www/html/site.dev/web
4.
5.     ServerName site.dev
6.
7.     <Directory "/var/www/html/site.dev/web">
8.         AllowOverride All
9.         Order deny,allow
10.        Require all granted
11.    </Directory>
12.
13.    ErrorLog ${APACHE_LOG_DIR}/site.dev-error.log
14.    CustomLog ${APACHE_LOG_DIR}/site.dev-access.log combined
15. </VirtualHost>
```

Figure 1.



Continue the installation following the steps shown in Figures 2 thru 7.

Figure 2.

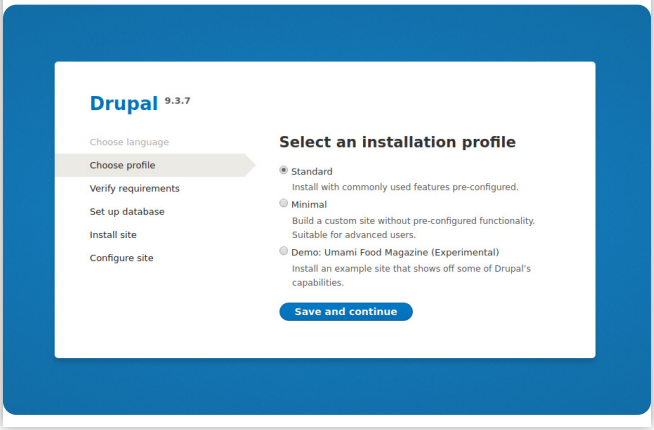


Figure 3 shows a red alert about the public directory and settings.php file. To resolve the problem, change the directory and file permissions. If you use Linux Ubuntu, type the following commands:

```
cd /var/www/html/site.dev/
mkdir web/sites/default/files
chown -R www-data:www-data web/sites/default/files/
cd web/sites/default/
cp default.settings.php settings.php
chown www-data:www-data settings.php
```

Figure 3.

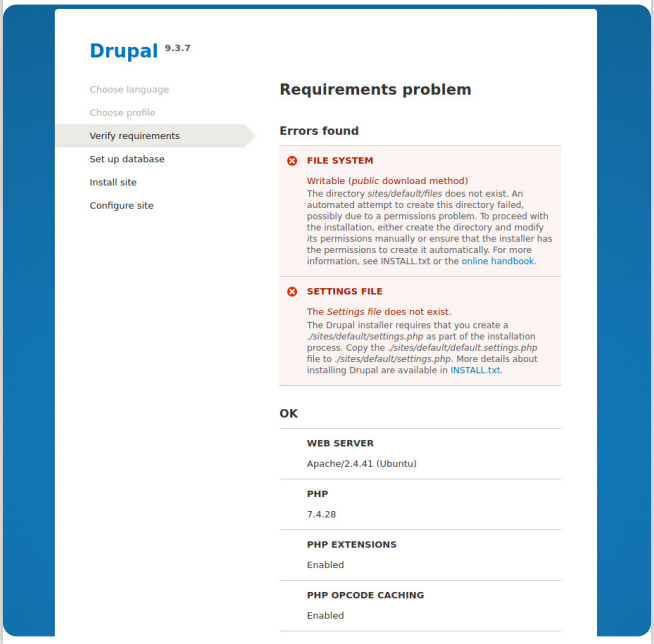
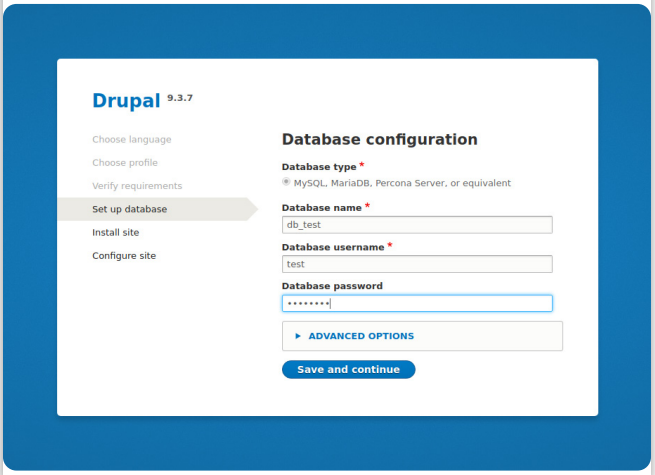


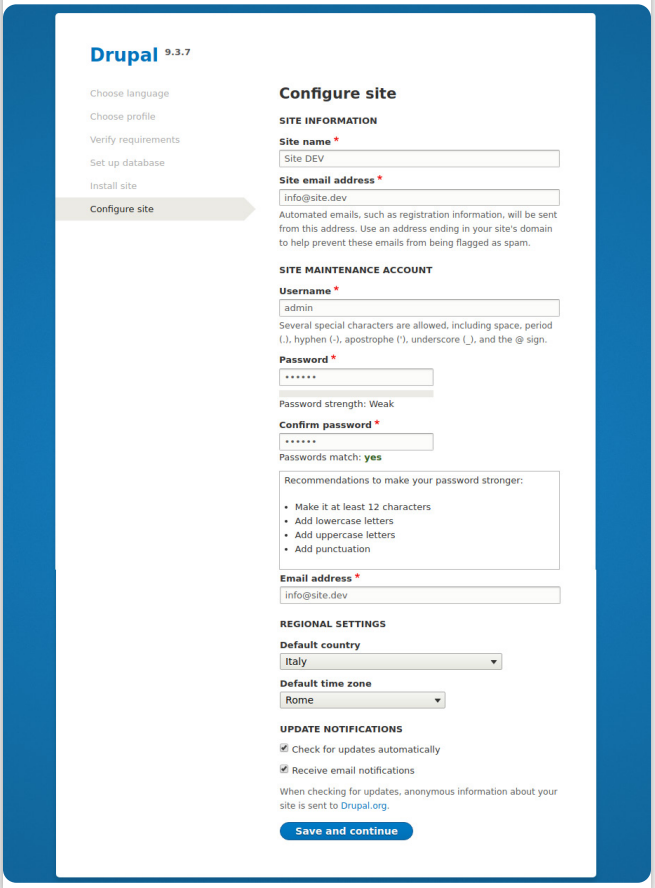
Figure 4 shows setting up the database — enter the database credentials.

Figure 4.



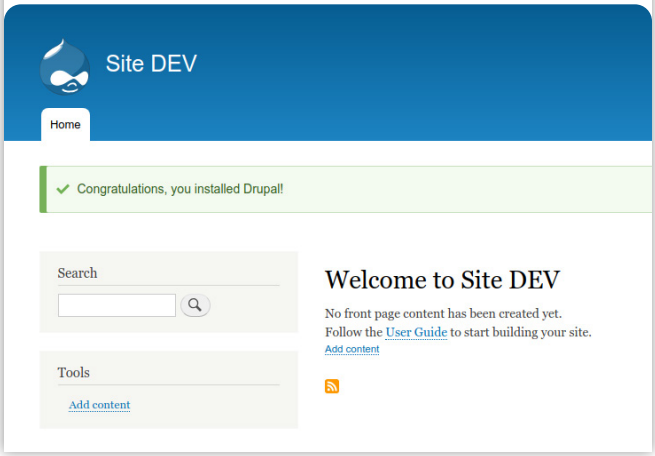
In Figure 5, we insert configuration site values. For example, administrator username, email to update, and administration.

Figure 5.



Finally, you see the final page.

Figure 6.



Install Drupal 9 Using the Command Line

You can use Drush to install Drupal from the command line. Add Drush to your project by running:

```
composer require drush/drush
```

Use `drush site:install` to run the command line set-up wizard. It'll install the standard profile without any arguments and ask only for database credentials.

```
drush site:install
```

To personalize the installation, use the following command to re-install using the expert install profile and set the default language to Italian.

```
drush site:install expert --locale=it
```

Install using the specified DB params:

```
drush site:install \
--db-url=mysql://test:test1234@localhost:3306/db_test
```

Re-install with the specified password for uid=1:

```
drush site:install --account-pass=mom
```

Installation based on the yml files stored in the config export/import directory:

```
drush site:install --existing-config
```

Disable email notification during install and later:

```
drush site:install standard \
install_configure_form.enable_update_status_emails=NULL
```

Download Drupal 9 Modules/themes Via Composer

Drupal contributed modules have dependencies on third-party libraries. If you install these modules/themes using Composer, it will download all the dependencies for you.

To download Drupal contributed modules or themes with Composer:

```
composer require drupal/<modulename>
composer require drupal/<themename>
```

For example, if you want to download the smtp module:

```
composer require drupal/smtp
```

The above command will be executed over the web directory, in our example `/var/www/html/site.dev`. Composer will automatically update your `composer.json` file, adding the module to all the other requirements in the list, like this:

```
{
  "require": {
    "drupal/smtp": "^1.5"
  }
}
```

You can enable the Drupal module in two ways. Using the standard Drupal web-browser interface or using a command-line tool like Drush:

```
drush en smtp
```

To download drush, run the following command:

```
composer require drush/drush
```

In all cases, Drupal also enables module dependencies.

Specifying a Version

You can specify the version of the module or theme that you want to download:

```
composer require 'drupal/<modulename>:<version>'
```

Cases are as follows:

- `composer require 'drupal/token:^1.5'` maps to the latest stable release [number drupal version].x-1.x release of the module:
- `composer require 'drupal/simple_fb_connect:~3.0'` maps to the latest stable release [number drupal version].x-3.x of the module.
- `composer require 'drupal/ctools:3.0.0-alpha26'` maps to version [number drupal version].x-3.0-alpha26.
- `composer require 'drupal/token:1.x-dev'` maps to [number drupal version].x-1.x-dev.

Composer will download the latest stable version if you don't specify the module or theme version.

You can download the dev version, but it is not recommended.

Search and Browser Module/theme

You can search for Drupal or non-Drupal projects from the command line using `composer search views`, which lists all projects that contain relations with views.

You can get info about the module/theme from the command line. The command, `composer search drupal/views`, opens a git page about the project into the browser.

Change the Default Installation Path

To change the locations where packages are installed, update the "installer-paths" section of the composer.json file. This approach uses the composer/installers package and uses a configuration such as shown in Listing 3.

In addition to the package type-based installation locations, you can use vendor-specific ones, like this:

```
"extra": {
    "installer-paths": {
        "web/libraries/ckeditor/plugins/{$name}": \
            ["vendor:ckeditor-plugin"]
    }
}
```

Or package-specific ones, like this:

```
"extra": {
    "installer-paths": {
        "web/libraries/{$name}": [ "enyo/dropzone" ]
    }
}
```

If a particular package matches multiple installer-path entries, the first one that matches will be used.

Updating Drupal 9 with Composer

To update the Drupal site using Composer you must first run `composer outdated "drupal/*"`, which lists available Drupal updates. If there is a line starting with drupal/core, continue with the commands below:

```
composer show drupal/core-recommended
```

This command verifies if the project uses drupal/core-recommended or drupal/core. If drupal/core-recommended is installed, it returns information about the package.

If drupal/core-recommended is not installed, it returns "Package drupal/core-recommended not found."

Core-recommended is tested and maintains all core dependencies at a fixed version for maximum stability. Drupal/core is useful when you want to use the latest dependencies and test that they work well.

If you are using drupal/core-recommended use the following command:

```
composer update drupal/core "drupal/core-*" \
 --with-all-dependencies
```

Listing 3.

```
1. "extra": {
2.     "installer-paths": {
3.         "core": ["type:drupal-core"],
4.         "libraries/{$name}": ["type:drupal-library"],
5.         "modules/contrib/{$name}": ["type:drupal-module"],
6.         "profiles/contrib/{$name}": ["type:drupal-profile"],
7.         "themes/contrib/{$name}": ["type:drupal-theme"],
8.         "drush/{$name}": ["type:drupal-drush"],
9.         "modules/custom/{$name}": ["type:drupal-custom-module"],
10.        "themes/custom/{$name}": ["type:drupal-custom-theme"]
11.    }
12. }
```

If you are not using drupal/core-recommended, but only drupal/core, use the following command:

```
composer update drupal/core --with-dependencies
```

To simulate the update without changing anything, use the following command.

```
composer update drupal/core --with-dependencies --dry-run
```

If there are updates, you can update using the web interface by visiting `http://site.dev/update.php` in your browser or with the command line using the following commands:

```
drush updatedb
drush cache:rebuild
```

You can't use Composer to see security updates; you must use drush.

```
drush pm:security
```

Updating Drupal 9 Into Production

If the above steps were performed on a dev/staging environment, to update an online production environment, copy composer.json and composer.lock files to production. Then run the following commands:

```
composer install --no-dev on production
drush updatedb
```

Notes About Drupal 9 Update

Before updating the site, make a backup of your files and database.

Always read the core release notes. Some contributed modules or themes may need updating to work with a new minor version of Drupal core. Patch releases shouldn't require this. To detect the needed module or theme updates, you need to read the project page or release notes.

Backup Database

To activate maintenance mode, use the following commands:

```
drush state:set system.maintenance_mode 1
drush cache:rebuild.
```

To dump the database, use the following command:

```
drush sql:dump
```

When finished, enable the site again with the following command:

```
drush state:set system.maintenance_mode 0
```

Backup File System

Traditional backup of file system is executed by tar or zip software.

```
tar cvfz backup.tgz \
 --exclude="/var/www/html/site.dev/vendor/" \
 /var/www/html/site.dev
```

You can use the "Backup & Migrate" module for alternative backup.

```
composer require drupal/backup_migrate
drush en backup_migrate
```

Update Drupal 9 Modules/themes with Composer

Commands for modules and themes are similar to Drupal core. The following command list available updates:

```
composer outdated "drupal/*" -->
```

This one lists available security updates:

```
drush pm:security --> List security updates
```

Use the following command for a minor-version update to a given Drupal module/project:

```
composer update drupal/modulename --with-dependencies
```

To preview the update without changing anything, add `--dry-run`.

```
composer update drupal/modulename --with-dependencies \
--dry-run
```

For a major version upgrade (such as 1.x to 2.x), execute the following steps:

- Require the new major version explicitly.

```
composer require drupal/modulename:^2.0
```

- Update module and dependencies:

```
composer update drupal/modulename --with-dependencies
```

Finally, execute the following commands.

```
drush updatedb
drush cache:rebuild
```

Conclusion

In this article, we talked about installing Drupal 9 through Composer, also mentioning the installation via a web interface. Since Drupal 8, Composer has become essential for the core installation and contributed modules/themes because it allows you to delegate the management of all dependencies of various modules. Imagine downloading and installing a module by hand, but after going to the modules page, you see that you need to download three other mandatory modules. Then after downloading them, you notice that they have additional dependencies. An endless nightmare! But by using Composer all this no longer matters. Composer will take care of downloading everything you need starting from the request of the first module. In addition to Drupal modules/themes, Composer will also download and configure third-party libraries.

Composer has many options, which you can see with the following command:

```
composer --help
```

If you want to see options about a command you must execute the following:

```
composer --help list
```

If you have any questions, you can contact nicola@pignatelli.com

Nicola Pignatelli has been building PHP applications since 2001 for many largest organizations in the field of publishing, mechanics and industrial production, startups, banking, and teaching. Currently, he is a Senior PHP Developer and Drupal Architect. Yes, this photo is of him.
@pignatellicom

Making Some Change

Oscar Merida

This article looks at a practical math problem most developers are likely to run into at some point. We're given some amount of money and asked to figure out how to distribute it. Did you remember that computers are not very good at floating-point math?

Recap

You're given a non-zero amount of dollars. Write a script that outputs the coins and dollar bill denominations that total the amount. The program must minimize the number of bills and coins used. For this exercise, assume you can use $100, $50, $20, $10, $5, and $1 dollar bills and quarters, nickels, dimes, and pennies.

For example, if the amount is $267.51, your script should output:

```
2 $100 bills
1 $50 bill
1 $10 bill
1 $5 bill
2 $1 bill
2 quarters
1 penny
```

The Naive Way

Your first attempt at a solution likely uses floats directly, as in Listing 1. The approach is logically correct. If we want to know how many $20 bills are in an amount, we divide `amount` by `$20.00` and use `floor()`[1] to get rid of any decimal remainders. We then reduce `amount` by `20.00` times the number of bills we need.

Listing 1.

```
1. $amount = 267.51;
2.
3. echo "Starting amount: {$amount}" . PHP_EOL;
4. if ($amount >= 100.00) {
5.     $hundreds = $amount / 100.00;
6.     echo floor($hundreds) . " $100 bill(s)" . PHP_EOL;
7.     $amount = $amount - (floor($hundreds) * 100.00);
8. }
9.
10. if ($amount >= 50.00) {
11.     $fifties = $amount / 50.00;
12.     echo floor($fifties) . " $50 bill(s)" . PHP_EOL;
13.     $amount = $amount - (floor($fifties) * 50.00);
14. }
15.
16. if ($amount >= 20.00) {
17.     $twenties = $amount / 20.00;
18.     echo floor($twenties) . " $20 bill(s)" . PHP_EOL;
19.     $amount = $amount - (floor($twenties) * 20.00);
20. }
21.
22. if ($amount >= 10.00) {
23.     $tens = $amount / 10.00;
24.     echo floor($tens) . " $10 bill(s)" . PHP_EOL;
25.     $amount = $amount - (floor($tens) * 10.00);
26. }
27.
28. if ($amount >= 5.00) {
29.     $fives = $amount / 5.0;
30.     echo floor($fives) . " $5 bill(s)" . PHP_EOL;
31.     $amount = $amount - (floor($fives) * 5.00);
32. }
33.
```

Listing 1 continued.

```
34. if ($amount >= 1.0) {
35.     $ones = $amount / 1.0;
36.     echo floor($ones) . " $1 bill(s)" . PHP_EOL;
37.     $amount = $amount - (floor($ones) * 1.00);
38. }
39.
40. if ($amount >= 0.25) {
41.     $quarters = $amount / 0.25;
42.     echo floor($quarters) . " quarter(s)" . PHP_EOL;
43.     $amount = $amount - (floor($quarters) * 0.25);
44. }
45.
46. if ($amount >= 0.10) {
47.     $nickels = $amount / 0.10;
48.     echo floor($nickels) . " nickels(s)" . PHP_EOL;
49.     $amount = $amount - (floor($nickels) * 0.10);
50. }
51.
52. if ($amount >= 0.05) {
53.     $dimes = $amount / 0.05;
54.     echo floor($dimes) . " dimes(s)" . PHP_EOL;
55.     $amount = $amount - (floor($dimes) * 0.05);
56. }
57.
58. if ($amount >= 0.01) {
59.     $pennies = $amount / 0.01;
60.     echo floor($pennies) . " pennies(s)" . PHP_EOL;
61.     $amount = $amount - (floor($pennies) * 0.01);
62. }
63.
64. echo "Remaining amount: " . $amount;
```

1 `floor()`: https://php.net/floor

Finally, we move on to the next smallest bill or coin that we have.

For many starting amounts, the code does output the correct breakdown of denominations and coins. But it does not work for all amounts, including the one given in our puzzle.

Looking at the output of our program—I have it echoing the starting and remaining amounts. When it reaches the code to calculate the number of pennies required, the value of our remaining amount value has somehow drifted to less than one cent. However, the drift is enough that it does not divide by `0.01`. Our solution is wrong.

```
Starting amount: 267.51
2 $100 bill(s)
1 $50 bill(s)
1 $10 bill(s)
1 $5 bill(s)
2 $1 bill
2 quarter(s)
Remaining amount: 0.0099999999999909
```

Converting to Integers

For consistency across input amounts, we must avoid using floating-point values. For any currency, we work with the smallest, indivisible unit of coinage that can be traded. In many cases, we can convert a two decimal number into an integer representing the number of hundredth units we have. For American dollars ($USD), we convert our starting number to an equivalent number of pennies. To do so, we multiply our starting amount (a float value) by `100` and cast it to an `int`.

```
$amount = (int) ($amount * 100);
```

The rest of the code (see Listing 2) requires a couple of changes:

- All the values for a denomination or coin are multiplied by 100 to express them as an equivalent number of pennies.
- We can use the modulus operator (`%`) to find the remainder. We use this remainder to check against the next smaller denomination we have.
- The pennies case collapses since any amount we have remaining when we get to that step can only be expressed as some number of pennies between 1 and 4.

Removing Repetitive Code

Our preceding solution is technically correct, but I see some repetitive code that I'd like to reduce. Moving chunks of similar code into reusable, abstract functions always pays off. Not only is the resulting code easier to read, but maintaining

Listing 2.

```
$amount = 267.51;
// convert to integer number of pennies
$amount = (int) floor($amount * 100);
// denominations
echo "Starting amount: {$amount}" . PHP_EOL;
if ($amount >= 10000) {
    $hundreds = (int) ($amount / 10000);
    echo (int) $hundreds . " $100 bill(s)" . PHP_EOL;
    $amount = $amount % 1000;
}

if ($amount >= 5000) {
    $fifties = (int) ($amount / 5000);
    echo (int) $fifties . " $50 bill(s)" . PHP_EOL;
    $amount = $amount  % 5000;
}

if ($amount >= 2000) {
    $twenties = (int) $amount / 2000;
    echo floor($twenties) . " $20 bill(s)" . PHP_EOL;
    $amount = $amount % 2000;
}

if ($amount >= 1000) {
    $tens = (int) ($amount / 1000);
    echo (int) $tens . " $10 bill(s)" . PHP_EOL;
    $amount = $amount % 1000;
}

if ($amount >= 500) {
    $fives = (int) ($amount / 500);
    echo $fives . " $5 bill(s)" . PHP_EOL;
    $amount = $amount % 500;
}

if ($amount >= 100) {
    $ones = (int) ($amount / 100);
    echo $ones . " $1 bill(s)" . PHP_EOL;
    $amount = $amount % 100;
}

if ($amount >= 25) {
    $quarters = (int) ($amount / 25);
    echo $quarters . " quarter(s)" . PHP_EOL;
    $amount = $amount % 25;
}

if ($amount >= 10) {
    $nickels = (int) ($amount / 10);
    echo $nickels . " nickels(s)" . PHP_EOL;
    $amount = $amount % 10;
}

if ($amount >= 5) {
    $dimes = (int) ($amount / 5);
    echo $dimes . " dimes(s)" . PHP_EOL;
    $amount = $amount % $dimes;
}

if ($amount >= 1) {
    echo $amount . " pennies(s)" . PHP_EOL;
    $amount = $amount - $amount; // 0
}
echo "Remaining amount: ". $amount;
```

it is simpler because any changes you need to make for a fix are in one place.

In Listing 3, I've added a function to perform our division and return both the quotient—the number of bills or coins—and the remaining amount to breakdown. To loop through the calculations, I define an array of the bills and coins in our cash registers. An immediate benefit of doing so is that we could, in the future, use this array to represent the state of a cash register and the actual bills and coins available for making change.

While lines-of-code is not a great measure of productivity or quality, this solution is 26 lines shorter than the previous one—about 2/3 as long. Removing the repeated code improves maintainability. If we have to make a change to it, we need to make the change in one place. If we had to make the same change on multiple lines of the code, we risk a typo or accidentally skipping one of the spots. Most of our time is spent reading and maintaining code, not writing new code. Crafting your code with an eye towards maintainability pays off in the long run.

Listing 3.

```
$amount = 267.51;

// convert to integer number of pennies
$amount = (int) ($amount * 100);

/**
 * @return array{quotient: int, remainder: int}
 */
function div(int $dividend, int $divisor): array {
    return [
        (int) ($dividend / $divisor),
        $dividend % $divisor
    ];
}

$denominations = [
    // [amount in pennies, string name, singular, plural
    [10000, '$100', 'bill', 'bills'],
    [5000, '$50', 'bill', 'bills'],
    [2000, '$20', 'bill', 'bills'],
    [1000, '$10', 'bill', 'bills'],
    [500, '$5', 'bill', 'bills'],
    [100, '$1', 'bill', 'bills'],
    [25, '', 'quarter', 'quarters'],
    [10, '', 'nickel', 'nickels'],
    [5, '', 'dime', 'dimes'],
    [1, '', 'penny', 'pennies'],
];

// denominations
echo "Starting amount: {$amount}" . PHP_EOL;

foreach ($denominations as $d) {
    if ($amount >= $d[0]) {
        [$num, $amount] = div($amount, $d[0]);
        echo $num . ' ' . $d[1] . ' '
            . ($num > 1 ? $d[3] : $d[2] ) . PHP_EOL;
    }
}

echo "Remaining amount: ". $amount;
```

Floating-Point Values

Working with floating-point values, also referred to as doubles or floats, are challenging for all computers and programming languages. It's not limited to PHP but a result of trying to represent base-10, fractional numbers in silicon registers built for binary operations. If you perform sensitive calculations with floats, consider using the functions from the bc_math[2] library that comes with PHP.

Working With Currency

If you take anything from this month's solution, remember that working with money is tricky. In this case, we're only working with one currency. Think about what might go wrong when you're converting between dollars and pounds, yen, or euros. How would you write code that has to handle transactions in multiple currencies without sacrificing accuracy? You would not want to rely on memory to convert floating-point values to integers throughout your code. Instead, use a Value Object to represent currency amounts for performing calculations. MoneyPHP[3] is a robust library from X that you can use in any PHP application. Read "Bug-free Money Handling with MoneyPHP"[4] to learn more about it.

Controlled Randomness

For next month, let's try to harness randomness such that we can guarantee that a given input will produce the same random sequence.

> *Given a positive integer, produce a sequence of six colors. The colors can be red, green, blue, yellow, purple, or black. The same integer input must always produce the same sequence of colors in the same order. Varying the input integer by one digit must produce a new sequence that is entirely different such that you can predict how the sequence changes when the integer input changes.*

Given `672985`, your code should consistently produce the same sequence. `green-purple-black-purple-black-purple`.

2 *bc_math:* https://php.net/bc_math

3 *MoneyPHP:* https://github.com/moneyphp

4 *"Bug-free Money Handling with MoneyPHP":* https://phpa.me/moneyphp

Some Guidelines And Tips

The puzzles can be solved with pure PHP. No frameworks or libraries are required.

- *Each solution is encapsulated in a function or a class, and I'll give a sample output to test your solution against.*
- *You're encouraged to make your first attempt at solving the problem without using the web for clues.*
- *Refactoring is encouraged.*
- *I'm not looking for speed, cleverness, or elegance in the solutions. I'm looking for solutions that work.*
- *Go ahead and try many solutions if you like.*
- *PHP's interactive shell (`php -a` at the command line) or 3rd party tools like PsySH[5] can be helpful when working on your solution.*
- *To keep solutions brief, we'll omit the handling of out-of-range inputs or exception checking.*

Related Reading

- *Bug-free Money Handling with MoneyPHP* by Piotr Horzycki, November 2020
  https://phpa.me/moneyphp-2020-11
- *Finding Prime Factors* by Oscar Merida, March 2022
  https://phpa.me/merdia-mar-2022
- *Calculating Population Growth* by Sherri Wheeler, September 2020
  https://phpa.me/wheeler-sept-2020

Oscar Merida has been working with PHP since version 4 was released and is constantly learning new things about it and still remembers installing an early version on Apache. When not coding or writing, he enjoys RPGs, soccer, and drawing. @omerida

5 *PsySH: https://psysh.org*

New and Noteworthy

PHP Releases

PHP 8.1.4 Released!:

https://www.php.net/archive/2022.php#2022-03-17-2

PHP 8.0.17 Released!:

https://www.php.net/archive/2022.php#2022-03-17-1

News

Symfony Live Paris 2022

SymfonyLive Paris returns in 2022! We are delighted to welcome you to SymfonyLive Paris 2022, the French-speaking event of Symfony!

https://live.symfony.com/2022-paris/

Platform.sh becomes the official Symfony PaaS

SymfonyCloud is a layer on top of Platform.sh[1] that adds features to make managing Symfony projects easier.

https://phpa.me/symfony-cloud-platform-sh

PhpStorm 2022.1 EAP #6

SymfonyLive Paris, the French Symfony conference welcomes the last conference speakers and their talks. Keynotes will be soon announced.

https://blog.jetbrains.com/?p=234950/

JetBrains' Statement on Ukraine

We have already made our position very clear in regard to the invasion of Ukraine. We condemn the attacks of the Russian government, and we stand with the Ukrainian people, including our own colleagues and their families.

https://blog.jetbrains.com/?p=233510/

PHP RFC: Arbitrary string interpolation

PHP has four types of string interpolation. The two most common forms are `$foo` and `{$foo}`. For details on the two other forms of string interpolation, see the deprecate ${} string interpolation RFC.

https://wiki.php.net/rfc/arbitrary_string_interpolation

Setup PHP in GitHub Actions

Setup PHP with required extensions, php.ini configuration, code-coverage support and various tools like composer in GitHub Actions.

https://github.com/shivammathur/setup-php

Generics in depth — Generics in PHP #2

This is the fourth video in a four-part series on using generics in PHP.

https://www.youtube.com/watch?v=208A9AgccKs

DrupalCon 2022 (PORTLAND, OR, USA / 25-28 APRIL 2022 / Oregon Convention Center)

Build your skills, learn more about what Drupal can do, and contribute to advancing the open source platform that organizations around the world rely on every day.

https://events.drupal.org/portland2022

1 *Platform.sh:*
http://platform.sh/

PSR 12 Extended Coding Style Standard

Frank Wallen

In last month's column of PSR Pickup, I talked about PSR-4:Autoloader (PSR-0 was the first Autoloader PSR) and PSR-1:Basic Coding Standard. In this month's column, I'll be talking about PSR-12:Extended Coding Style Standard. PSR-12 builds on PSR-1 and is significant in the number of rules it defines, most of which are formatting styles well-supported by modern IDEs.

Modern IDEs have alleviated most of the styling expected from the programmer, cleaning up with a simple keystroke or mouse-click. For that reason, I will focus more on the rules that, in my opinion, have more "real world" implications. What I mean by that are those particular rules similar to PSR-1's rule about Byte Order Marking (BOM) in a file that can produce real-world bugs. Whereas whether we use `StudlyCaps` or `snake_case` for class names has much more to do with style and communication between developers (not that communication is not important, of course) than the functionality of our scripts. However, it's also important to follow the standards defined and expected in one's team. Not everyone follows all PSR rules exactly. The needs of technology, architecture, legacy, etc., and the team preferences must be considered. In my experience, teams working in legacy applications opt to follow the original style more closely, placement of braces, variable name casing (i.e., `snake_case` vs. `camelCase`) in the scripts where changes are made (corrections or new code). Often, the expectation is not to make a commit to the repo strictly for style changes; many teams feel it is not a good use of time for the developer and PR reviewers.

This PSR supports and expands on PSR-1. An excellent example of the evolving nature of PSRs is PSR-12 which replaced and deprecated PSR-2:Coding Style Guide. PSR-2 was accepted in 2012, but there have been numerous changes to the PHP language since then, and adopting it could result in a large amount of work. From the meta-document for PSR-12:

> *PSR-2 was created based upon the common practices of the PHP-FIG projects at the time but ultimately this meant it was a compromise of many of the different projects' guidelines. The repercussions of projects changing their coding guidelines to align with PSR-2 (Almost all projects do align with PSR-1, even if it is not explicitly stated) were seen to be too great (losing git history, huge changesets, and breaking existing patches/pull requests).*

Figure 1.

```
...  ...   @@ -0,0 +1,4 @@
          1 + <?php
          2 +
          3 + $options = [];
          4 + $required = [];
```

All PHP files MUST use the Unix LF (linefeed) line ending only.

All PHP files MUST end with a non-blank line, terminated with a single LF.

These two rules seem almost inconsequential, don't they? In terms of being compiled for execution, they are inconsequential—no impact on the execution of your code. However, what can be impacted are the tools used in your development or deployment pipelines. Some tools only recognize a line that ends in a LF, so they actually might ignore your last line of code or have trouble diffing it. If there's no LF on the last line and another developer must add lines after that, they must add a LF before their own lines, meaning that now a diff tool might think that developer wrote the previous line, the one that originally didn't end in a LF. In Figure 1, we can see that we do not have a LF at the end of line 4:

Now imagine a second developer is adding some code following line 4. The developer would need to add a LF to line 4, then add their line on 5. One can clearly see that line 4 was 'replaced' and line 5 added, but that really isn't the case; only the LF was added to the end of line 4.

Figure 2.

```
...  ...   @@ -1,4 +1,5 @@
   1    1   <?php
   2    2
   3    3   $options = [];
   4        - $required = [];
        4   + $required = [];
        5   + $additional = [];
```

Granted, this isn't terrible, and luckily most IDEs will automatically add the LF for you.

The closing ?> tag MUST be omitted from files containing only PHP.

One reason for this is similar to the BOM in a file. When a script is imported using `require` or `include` and there happens to be a space after `?>`, it could trigger output in the including file before it has properly emitted its headers, resulting in the dreaded `Headers already sent` error. It might also add unwanted whitespace during output. In the browser, that is less likely to create a problem as the space is invisible and doesn't impact the index like a non-breaking space ( ). But if a download file is generated (like a TAB file), then that could easily introduce unexpected behavior in the consumer of that file.

Code MUST use an indent of 4 spaces for each indent level, and MUST NOT use tabs for indenting.

This rule is a classic long-running debate among developers. Luckily, an IDE will usually allow you to set the number of spaces representing a tab, so you can just strike the TAB key easily, and the IDE inserts spaces. Mixing spaces with tabs can be handled elegantly by an IDE, leaving the user unaffected as the code looks properly formatted. But version control software, like Git, handles whitespace differently. A `space` followed by a `TAB` in an IDE appears as if it were only a single `TAB`, but version control would see the difference and consider it a change. While not impacting the functionality of the code, it can present as cognitive friction when another developer is reviewing the code.

Figure 3 shows TAB characters used instead of spaces and a space added before the TAB. There is no visual difference in the IDE and no real issue.

Figure 3.

```
for($idx = 1; $idx <= 10; $idx++) {
    $a = $idx/2;
    $b = $a + 1;
}
```

The screenshot in Figure 4 is taken from a diff on Github. Note Line 10 now shows a space added before the tab and the line is considered changed.

Figure 4.

Again, this isn't terrible either, but it highlights the potential confusion of mixing spaces and tabs. While you may not see it in the IDE, the diff tool noticed.

Conclusion

I have only highlighted the rules from PSR-12 that I believe can impact the developer's experience when it comes to tooling and collaboration. There are many more rules in PSR-12[1] that I recommend reviewing. Discuss them with your team to decide which ones to follow, and write up your own style guide for new members. If it's in the team style guide, it's established, and development can continue. This up-front work eliminates wasting time and effort debating whether a rule from PSR-12 should be implemented. Admittedly, styling PSRs are not as impactful as other PSRs, but they are significant in terms of collaboration and should be considered just as important.

Frank Wallen is a PHP developer and tabletop gaming geek (really a gamer geek in general). He is also the proud father of two amazing young men and grandfather to two beautiful children who light up his life. He lives in Southern California and hopes to one day have a cat again. @frank_wallen

Related Reading

- *PSRs - Improving the Developer Experience* by Frank Wallen, March 2022. https://phpa.me/wallen-mar-2022
- *Mentoring and Teaching PHP* by Ken Marks, July 2021 https://phpa.me/teaching-php-2021
- *Working with PHP Streams* by Chris Tankersley, January 2021 http://phpa.me/education-jan-21

1 *PSR-12: https://www.php-fig.org/psr/psr-12/*

Tech is Taking Sides

Beth Tucker Long

Throughout history, industries have stayed relatively neutral during wartime. Global companies, especially, may offer marketing-focused messages of hope and concern but keep their heads down and their tones neutral when faced with actually taking a stand against one side of a conflict. Per usual, though, the tech industry is happy to disrupt the status quo—not just taking a clear stand but putting their money and their talent where their mouth is.

Global industries have long been accused of profiting off of war by selling to all sides, especially those in manufacturing and raw materials. Tech companies around the world, though, are taking a defined stand and even putting aside opportunities to profit as they take sides in the current conflict between Ukraine and Russia.

SpaceX has donated Starlink Internet communications systems to keep people in Ukraine connected to the internet. AT&T, Verizon, T-Mobile, and Vodafone have all waived fees for services involving Ukraine—some waiving roaming charges for customers in Ukraine, some waiving international call fees for calls to and from Ukraine. Apple has halted all sales through its Russian online store, shut down Apply Pay access in Russia, and has disabled traffic and incident reports within Ukraine in Apple Maps. Google, likewise, has disabled live traffic reports for Ukraine in Google Maps, blocked Russian state-sponsored media channels on YouTube, and paused ad sales in Russia. Microsoft has stopped sales to Russia, and even Netflix has gotten involved, suspending their services in Russia.

Even companies that are trying to just carry on with business-as-usual are having trouble staying uninvolved. Despite not taking a public stand in the matter, Coinbase, Patreon, and many other fundraising, financial, and social media platforms have found themselves in the spotlight as their platforms are being used heavily to support various organizations involved in the Russia-Ukraine conflict. It is no longer easy to keep your head down and stay neutral.

Outside of work, the tech community is also getting involved. Over 300,000 programmers have volunteered to become an "IT Army" on behalf of Ukraine, taking down key Russian websites and online services through coordinated DDoS attacks, reporting disinformation on social media platforms, and creating digital marketing campaigns to bring awareness to the conflict. Anonymous has claimed credit for hacking Russia's television networks to broadcast footage from Ukraine and for accessing the corporate systems of Rosatom, Russia's state corporation for nuclear energy, after they claimed ownership of a Ukrainian nuclear power plant.

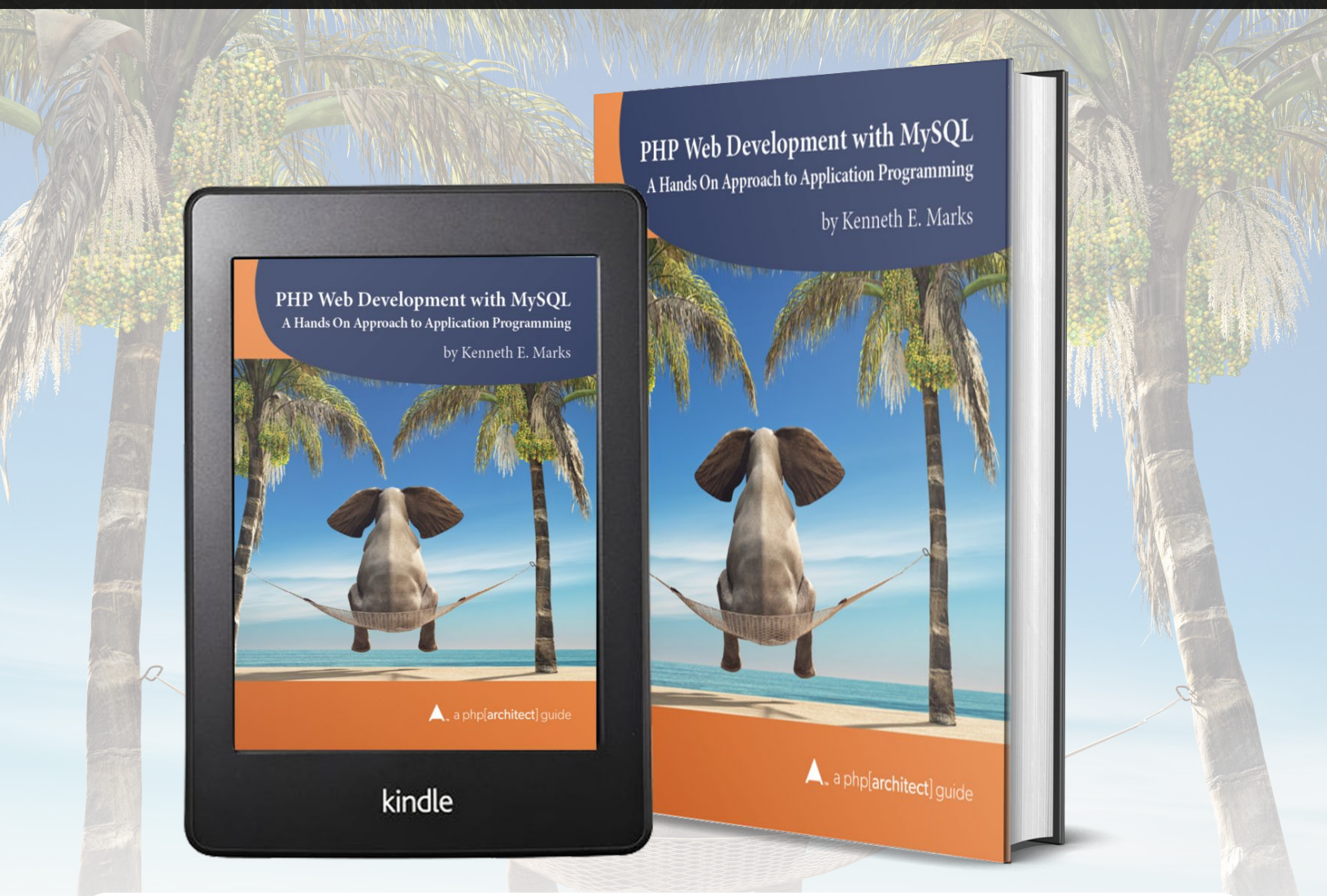
Regardless of the outcome of this conflict, it is no longer isolated and localized. People around the world are getting involved. Strangers are fighting together while sitting in their living rooms halfway around the world from each other. One thing is abundantly clear—tech has drastically changed the landscape of war.

Beth Tucker Long is a developer and owner at Treeline Design, a web development company, and runs Exploricon, a gaming convention, along with her husband, Chris. She leads the Madison Web Design & Development and Full Stack Madison user groups. You can find her on her blog (http://www.alittleofboth.com) or on Twitter @e3BethT

Related URLs

- *US Carriers Waive Fees for Customers Who Need to Call Ukraine* https://phpa.me/cnet-mobile-carrier-support
- *Apple Halts All Sales From Online Store in Russia* https://phpa.me/macrumors-apple-stop-sales
- *YouTube is now blocking Russian state-funded media worldwide* https://phpa.me/theverge-youtubeblocks-russia
- *Elon Musk Warns of Russian Attacks on Donated Starlink Internet Hubs in Ukraine* https://phpa.me/cnet-donated-starlink-hubs
- *Corporations and Big Tech Find Ways to Help Ukraine* https://phpa.me/voanews-tech-helps-ukraine
- *Ukraine crisis: Google Maps live traffic data turned off in country* https://www.bbc.com/news/technology-60561089
- *'It's the right thing to do': the 300,000 volunteer hackers coming together to fight Russia* https://www.theguardian.com/p/yxf9p
- *Anonymous hacks Russian firm running Ukrainian nuclear plant* https://www.jpost.com/breaking-news/article-701402
- *Netflix suspends its service in Russia as Western companies take stand over Ukraine invasion* https://phpa.me/netflix-suspends-russia
- *On the removal of Come Back Alive* https://blog.patreon.com/on-the-removal-of-come-back-alive
- *Coinbase Is Latest Exchange to Deny Ukraine Request to Block Russian Crypto Users* https://phpa.me/decrypt-coinbase-declines-block

a php[architect] guide



Learn how to build dynamic and secure websites.

The book also walks you through building a typical Create-Read-Update-Delete (CRUD) application. Along the way, you'll get solid, practical advice on how to add authentication, handle file uploads, safely store passwords, application security, and more.

Available in Print+Digital and Digital Editions.

Purchase Your Copy

https://phpa.me/php-development-book