

Test Anywhere Protocol

by Jeremiah Foster
jeremiah@jeremiahfoster.com



“Unit testing is not optional because working software is not optional.”—From a job posting on craigslist (<http://portland.craigslist.org/wsc/sof/650625710.html>).

■ What good is testing? -----

Why should I care about testing? Well I shouldn't, unless, of course, I want my software to work. If I hack a little code, run the code to check its output, then I'm testing. So why not preserve those small tests in code and run it against my new module or program?

If I do that, I find that I catch a lot of unexpected bugs and learn more about how things work. Testing is an integral part of Perl, as anyone who has programmed in Perl for a while will tell me. When I add a new module from CPAN and I see all those “ok” messages shoot by, those are from various tests built into the module I'm downloading. In fact the “ok” is the fundamental unit of Perl testing showing how simple the testing syntax of Perl is.

When I contribute to CPAN, my software enters a matrix of testing; from the CPAN Testing Service (CPANTS, <http://cpants.perl.org>), to tests measuring “kwalitee”, to CPAN testers (<http://testers.cpan.org>), to end users who will leave reviews on CPAN ratings (<http://ratings.cpan.org>)—my code is very public and well tested.

Of course, I may be a seasoned software designer, forced by employers and habit to run tests. I want to run my tests in an automated fashion but I don't want to sift through the voluminous output of the test log afterwards—I want just the juicy bits; did my tests pass or fail?

■ Test Anywhere Protocol -----

Whether I'm a n00b or seasoned vet, TAP is the perfect thing for my toolkit. TAP is the “Test Anything Protocol” designed to run my tests and report back to me the relevant info I want—and only that. It is flexible enough to give me a complete picture of what is going on, but terse enough not to weigh me down with a novel thick enough to stun an ox every time I run my tests. TAP is a reporting layer between me and my tests.

This reporting layer is as agnostic as it gets: it does not care what language my test is written in as long as my program produces proper TAP output. So I can use the best tool for the job at hand. I can use a variety of programming languages and mix them together and TAP will still be able to check and report back the output.

In fact, I'm going to use both Perl and shell scripts (bash)

to demonstrate TAP in this article showing how flexible it can be. TAP is useless of course without a test to report on, so I'll create my first test.

Starting with something simple for the purpose of this article, I'll write a Perl script that checks which operating system I'm using. This type of test might be useful for my Perl script since resources can be located in different places depending on operating system.

To get the operating system, I'm going to use Perl's built in variable `$^O`. Using Perl's built-in variable for the OS helps ensure that I'm going to be as cross-platform as possible. Since there are Perl porters and other wizards working to ensure Perl works on as much hardware as possible, let them do the heavy lifting for us.

My test is in Code Listing 1 (next page). This simple script is all I need to get started. Note that I use warnings, `-w` in the shebang line, and `strict` (the `use strict` line.) I added two comment lines describing the script, then the last two lines create the output, on my machine it prints the text in *Output Listing 1* (next page):

Output Listing 1 is TAP output. I know it is TAP output because I used the module `Test::More` which is known as a TAP producer; there are other TAP producers on CPAN. The first line of the output reflects the number of tests I said I would run when I called `Test::More`.

The second and third lines are the results of those individual tests; first, I see the “ok” and then I see a number indicating the test and any description I gave. The last line is a bonus; it is TAP trying to reconcile the fact that I told it I would run 3 tests but I ended up only running 2. This type of diagnostic message is invaluable as I will see later: it allows me to pinpoint the problem with my code. I'll add a line to test my Perl version:

```
is($], 5.008008, 'Perl version 5.8.8');
```

When I run my test again (still using Perl 5.8.8) I get the text in *Output Listing 2* (next page).

Now that I have fixed my test script, it is time to incorporate some TAP reporting. One incredibly useful TAP parser is located in the `Test::Harness` module. To get it, whatever method is available for my operating system, for example I recommend using Strawberry Perl (<http://www.strawberryperl.com>) and normal CPAN module installation if I'm using Windows. Since I prefer debian for my operating system, I can just use a command-line tool:

```
aptitude install libtest-harness-perl
```