Home
Testing with TAP
Producers
Consumers

TAP specification

NAME

TAP - The Test Anything Protocol

SYNOPSIS

TAP, the Test Anything Protocol, is Perl's simple text-based interface between testing modules such as Test::More and a test harness such as Test::Harness 2.x or TAP::Harness 3.x.

THE TAP FORMAT

TAP's general format is:

```
1..N
ok 1 Description # Directive
# Diagnostic
....
ok 47 Description
ok 48 Description
more tests....
```

For example, a test file's output might look like:

```
1..4
ok 1 - Input file opened
not ok 2 - First line of the input valid
ok 3 - Read the rest of the file
not ok 4 - Summarized correctly # TODO Not written yet
```

HARNESS BEHAVIOR

In this document, the "harness" is any program analyzing TAP output. Typically this will be Perl's prove program, or the underlying Test::Harness::runtests subroutine. A harness must only read TAP output from standard output and not from standard error. Lines written to standard output matching /^(not)?ok\b/ must be interpreted as test lines. All other lines must not be considered test output.

TESTS LINES AND THE PLAN

The version

The current version of TAP is 12. To allow future enhancements any future version greater than 12 should start with an explicit version line.

```
TAP version 13
```

In the absence of any version line version 12 is assumed. It is an error to explicitly specify any version lower than 13.

The plan

The plan tells how many tests will be run, or how many tests have run. It's a check that the test file hasn't stopped prematurely. It must appear once, whether at the beginning or end of the output. The plan is usually the first line of TAP output (although in future there may be a version line before it) and it specifies how many test points are to follow. For example,

```
1..10
```

means you plan on running 10 tests. This is a safeguard in case your test file dies silently in the middle of its run. The plan is optional but if there is a plan before the test points it must be the first non-diagnostic line output by the test file. In certain instances a test file may not know how many test points it will ultimately be running. In this case the plan can be the last non-diagnostic line in the output. The plan cannot appear in the middle of the output, nor can it appear more than once.

The test line

The core of TAP is the test line. A test file prints one test line test point executed. There must be at least one test line in TAP output. Each test line comprises the following elements:

- ok or not ok

This tells whether the test point passed or failed. It must be at the beginning of the line. /^not ok/ indicates a failed test point. /^ok/ is a successful test point. This is the only mandatory part of the line. Note that unlike the Directives below, ok and not ok are case-sensitive.

- Test number

TAP expects the ok or not ok to be followed by a test point number. If there is no number the harness must maintain its own counter until the script supplies test numbers again. So the following test output

```
1..6
not ok
ok
not ok
ok
ok
```

has five tests. The sixth is missing. Test::Harness will generate

```
FAILED tests 1, 3, 6
Failed 3/6 tests, 50.00% okay
```

- Description

Any text after the test number but before a # is the description of the test point.

```
ok 42 this is the description of the test
```

Descriptions should not begin with a digit so that they are not confused with the test point number. The harness may do whatever it wants with the description.

- Directive

The test point may include a directive, following a hash on the test line. There are currently two directives allowed: TODO and SKIP. These are discussed below.

To summarize: - ok/not ok (required) - Test number (recommended) - Description (recommended) - Directive (only when necessary)

DIRECTIVES

Directives are special notes that follow a # on the test line. Only two are currently defined: TODO and SKIP. Note that these two keywords are not case-sensitive.

TODO tests

If the directive starts with # TODO, the test is counted as a todo test, and the text after TODO is the explanation.

```
not ok 13 # TODO bend space and time
```

Note that if the TODO has an explanation it must be separated from TODO by a space. These tests represent a feature to be implemented or a bug to be fixed and act as something of an executable "things to do" list. They are **not** expected to succeed. Should a todo test point begin succeeding, the harness should report it as a bonus. This indicates that whatever you were supposed to do has been done and you should promote this to a normal test point.

Skipping tests

If the directive starts with # SKIP, the test is counted as having been skipped. If the whole test file succeeds, the count of skipped tests is included in the generated output. The harness should report the text after # SKIP\S*\s+ as a reason for skipping.

```
ok 23 # skip Insufficient flogiston pressure.
```

Similarly, one can include an explanation in a plan line, emitted if the test file is skipped completely:

```
1..0 # Skipped: WWW::Mechanize not installed
```

OTHER LINES

Bail out!

As an emergency measure a test script can decide that further tests are useless (e.g. missing dependencies) and testing should stop immediately. In that case the test script prints the magic words

```
Bail out!
```

to standard output. Any message after these words must be displayed by the interpreter as the reason why testing must be stopped, as in

```
Bail out! MySQL is not running.
```

Diagnostics

Additional information may be put into the testing output on separate lines. Diagnostic lines should begin with a #, which the harness must ignore, at least as far as analyzing the test results. The harness is free, however, to display the diagnostics. Typically diagnostics are used to provide information about the environment in which test file is running, or to delineate a group of tests.

```
...
ok 18 - Closed database connection
# End of database section.
# This starts the network part of the test.
# Daemon started on port 2112
ok 19 - Opened socket
...
ok 47 - Closed socket
# End of network tests
```

Anything else

Any output line that is not a version, a plan, a test line, a diagnostic or a bail out is considered an "unknown" line. A TAP parser is required to not consider an unknown line as an error but may optionally choose to capture said line and hand it to the test harness, which may have custom behavior attached. This is to allow for forward compatability. Test::Harness silently ignores incorrect lines, but will become more stringent in the future. TAP::Harness reports TAP syntax errors at the end of a test run.

EXAMPLES

All names, places, and events depicted in any example are wholly fictitious and bear no resemblance to, connection with, or relation to any real entity. Any such similarity is purely coincidental, unintentional, and unintended.

Common with explanation

The following TAP listing declares that six tests follow as well as provides handy feedback as to what the test is about to do. All six tests pass.

```
1..6
#
# Create a new Board and Tile, then place
# the Tile onto the board.
#
```

```
ok 1 - The object isa Board
ok 2 - Board size is zero
ok 3 - The object isa Tile
ok 4 - Get possible places to put the Tile
ok 5 - Placing the tile produces no error
ok 6 - Board size is 1
```

Unknown amount and failures

This hypothetical test program ensures that a handful of servers are online and network-accessible. Because it retrieves the hypothetical servers from a database, it doesn't know exactly how many servers it will need to ping. Thus, the test count is declared at the bottom after all the test points have run. Also, two of the tests fail.

```
ok 1 - retrieving servers from the database
# need to ping 6 servers
ok 2 - pinged diamond
ok 3 - pinged ruby
not ok 4 - pinged saphire
ok 5 - pinged onyx
not ok 6 - pinged quartz
ok 7 - pinged gold
1..7
```

Giving up

This listing reports that a pile of tests are going to be run. However, the first test fails, reportedly because a connection to the database could not be established. The program decided that continuing was pointless and exited.

```
1..573
not ok 1 - database handle
Bail out! Couldn't connect to database.
```

Skipping a few

The following listing plans on running 5 tests. However, our program decided to not run tests 2 thru 5 at all. To properly report this, the tests are marked as being skipped.

```
1..5
ok 1 - approved operating system
# $^0 is solaris
```

```
ok 2 - # SKIP no /sys directory
ok 3 - # SKIP no /sys directory
ok 4 - # SKIP no /sys directory
ok 5 - # SKIP no /sys directory
```

Skipping everything

This listing shows that the entire listing is a skip. No tests were run.

```
1..0 # skip because English-to-French translator isn't installed
```

Got spare tuits?

The following example reports that four tests are run and the last two tests failed. However, because the failing tests are marked as things to do later, they are considered successes. Thus, a harness should report this entire listing as a success.

```
1..4
ok 1 - Creating test program
ok 2 - Test program runs, no error
not ok 3 - infinite loop # TODO halting problem unsolved
not ok 4 - infinite loop 2 # TODO halting problem unsolved
```

Creative liberties

This listing shows an alternate output where the test numbers aren't provided. The test also reports the state of a ficticious board game in diagnostic form. Finally, the test count is reported at the end.

```
ok - created Board
ok
ok
ok
ok
ok
ok
ok
# +------+------+------+------+
# |      |16G   |      |05C   |
# |      |G N C |      |C C G |
# |      |  G   |      |  C  +|
# +------+------+------+------+
# |10C   |01G   |      |03C   |
# |R N G |G A G |      |C C C |
# |  R   |  G   |      |  C  +|
# +------+------+------+------+
# |      |01G   |17C   |00C   |
# |      |G A G |G N R |R N R |
# |      |  G   |  R   |  G   |
# +------+------+------+------+
ok - board has 7 tiles + starter tile
1..9
```

Non-Perl TAP

In Perl, we use Test::Simple and Test::More to generate TAP output. Other languages have solutions that generate TAP, so that they can take advantage of Test::Harness. The following sections are provided by their maintainers, and may not be up-to-date.

C/C++

libtap makes it easy to write test programs in C that produce TAP-compatible output. Modeled on the Test::More API, libtap contains all the functions you need to:

- Specify a test plan
- Run tests
- Skip tests in certain situations
- Have TODO tests
- Produce TAP compatible diagnostics

More information about libtap, including download links, checksums, anonymous access to the Subersion repository, and a bug tracking system, can be found at:

```
http://jc.ngo.org.uk/trac-bin/trac.cgi/wiki/LibTap
```

(Nik Clayton, April 17, 2006)

Python

PyTap will, when it's done, provide a simple, assertive (Test::More-like) interface for writing tests in Python. It will output TAP and will include the functionality found in Test::Builder and Test::More. It will try to make it easy to add more test code (so you can write your own TAP.StringDiff, for example. Right now, it's got a fair bit of the basics needed to emulate Test::More, and I think it's easy to add more stuff -- just like Test::Builder, there's a singleton that you can get at easily. I need to better identify and finish implementing the most basic tests. I am not a Python guru, I just use it from time to time, so my aim may not be true. I need to write tests for it, which means either relying on Perl for the tester tester, or writing one in Python. Here's a sample test, as found in my Subversion:

```
from TAP.Simple import *

plan(15)

ok(1)
ok(1, "everything is OK!")
ok(0, "always fails")

is_ok(10, 10, "is ten ten?")
is_ok(ok, ok, "even ok is ok!")
ok(id(ok),    "ok is not the null pointer")
ok(True,      "the Truth will set you ok")
ok(not False, "and nothing but the truth")
ok(False,     "and we'll know if you lie to us")

isa_ok(10, int, "10")
isa_ok('ok', str, "some string")

ok(0,    "zero is true", todo="be more like Ruby!")
ok(None, "none is true", skip="not possible in this universe")

eq_ok("not", "equal", "two strings are not equal");
```

(Ricardo Signes, April 17, 2006)

JavaScript

Test.Simple looks and acts just like TAP, although in reality it's tracking test results in an object rather than scraping them from a print buffer.

```
http://openjsan.org/doc/t/th/theory/Test/Simple/
```

(David Wheeler, April 17, 2006)

PHP

All the big PHP players now produce TAP

- phpt

Outputs TAP by default as of the yet-to-be-released PEAR 1.5.0

```
http://pear.php.net/PEAR
```

- PHPUnit

Has a TAP logger (since 2.3.4)

```
http://www.phpunit.de/wiki/Main_Page
```

- SimpleTest There's a third-party TAP reporting extension for SimpleTest

```
http://www.digitalsandwich.com/archives/51-Updated-Simpletest+Apache-Test.html
```

- Apache-Test

Apache-Test's PHP writes TAP by default and includes the standalone test-more.php

```
http://search.cpan.org/dist/Apache-Test/
```

(Geoffrey Young, April 17, 2006)

Java

tap4j provides objects to create TestSets, TestResults, BailOuts and, using a TapProducer, dump it into a java.io.File or String. It also provides TapConsumer's to load TapStreams from Strings or java.io.File's. YAMLish is interpreted and made available as diagnostics of each TestResult object of a TestSet.

- tap4j
- Example of a TapConsumer in Java with tap4j + YAMLish
- Example of a TapProducer in Java with tap4j + YAMLish

tap4j is licensed under MIT License.

BUGS

Please report any bugs or feature requests to bug-tap at rt.cpan.org, or through the web interface at http://rt.cpan.org/NoAuth/ReportBug.html?Queue=TAP. I will be notified, and then you'll automatically be notified of progress on your bug as I make changes.

SUPPORT

You can find documentation for this module with the perldoc command.

```
perldoc TAP
```

You can also look for information at:

- AnnoCPAN: Annotated CPAN documentation http://annocpan.org/dist/TAP
- CPAN Ratings http://cpanratings.perl.org/d/TAP
- RT: CPAN's request tracker http://rt.cpan.org/NoAuth/Bugs.html?Dist=TAP
- Search CPAN http://search.cpan.org/dist/TAP

AUTHORS

Andy Lester, based on the original Test::Harness documentation by Michael Schwern.

ACKNOWLEDGEMENTS

Thanks to Pete Krawczyk, Paul Johnson, Ian Langworth and Nik Clayton for help and contributions on this document. The basis for the TAP format was created by Larry Wall in the original test script for Perl 1. Tim Bunce and Andreas Koenig developed it further with their modifications to Test::Harness.

COPYRIGHT

TODO

Define the exit code of the process.

github.com/testanything