PHPUnit support in PhpStorm

In this tutorial we will have a look at PHPUnit support in PhpStorm

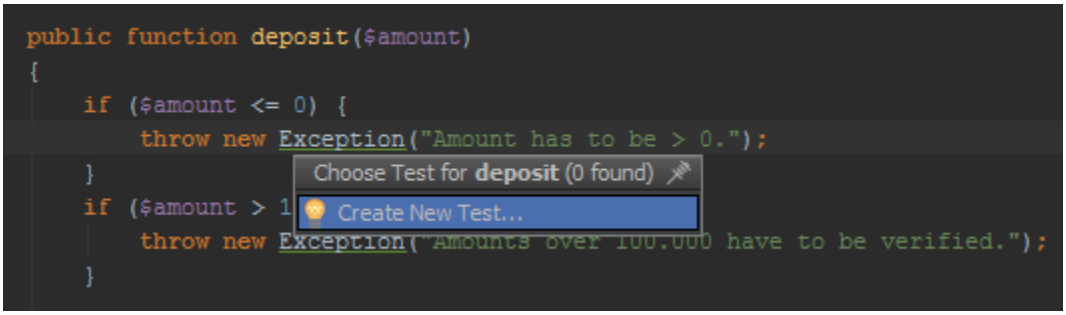
With unit testing, we can verify parts of our source code are working as expected. After we've changed our code or performed a refactoring, unit tests can tell us if the changes we did break existing functionality or not. Only when all the tests are "green" (all tests pass) can we be sure that we're not breaking the functionality of our code.

PhpStorm uses PHPUnit as the test runner, a widely used unit testing framework for PHP which provides a lot of features. Let's see how it integrates with PhpStorm.

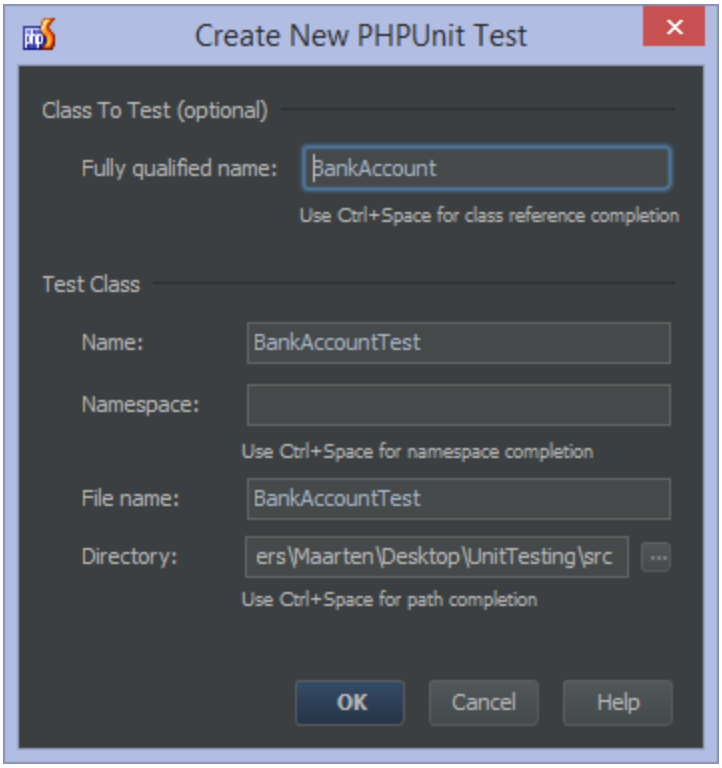
- 1. Adding a unit test to a project
- 2. Enabling PHPUnit for our project
- 3. Running unit tests
- 4. Reviewing test results

1. Adding a unit test to a project

Adding unit tests to a project can be done in several ways. We can create a new file and select the PHPUnit | PHPUnit test file template. Another way to create a test for some specific class is to invoke use the Go to Test action (with Navigate | Go to Test or Cmd+Shift+T / Ctrl+Shift+T) and choose Create new test.



Either method for creating a new test will open Create New PHPUnit Test dialog, in which we can specify the name of the class to test, the name and namespace of the test class, and where the PHP file should be saved.



After clicking OK, a boilerplate unit test class will be generated.

```php
<?php
class BankAccountTest extends PHPUnit_Framework_TestCase {

}
```

For more info on creating unit tests, see creating PHPUnit Tests in PhpStorm.

2. Enabling PHPUnit for our project

As we can see from the generated class above, there is no autocompletion support yet and PhpStorm does not know about the PHPUnit_Framework_TestCase class PHPUnit provides. Why is that? Because we haven't enabled PHPUnit yet for our project. Let's do that.

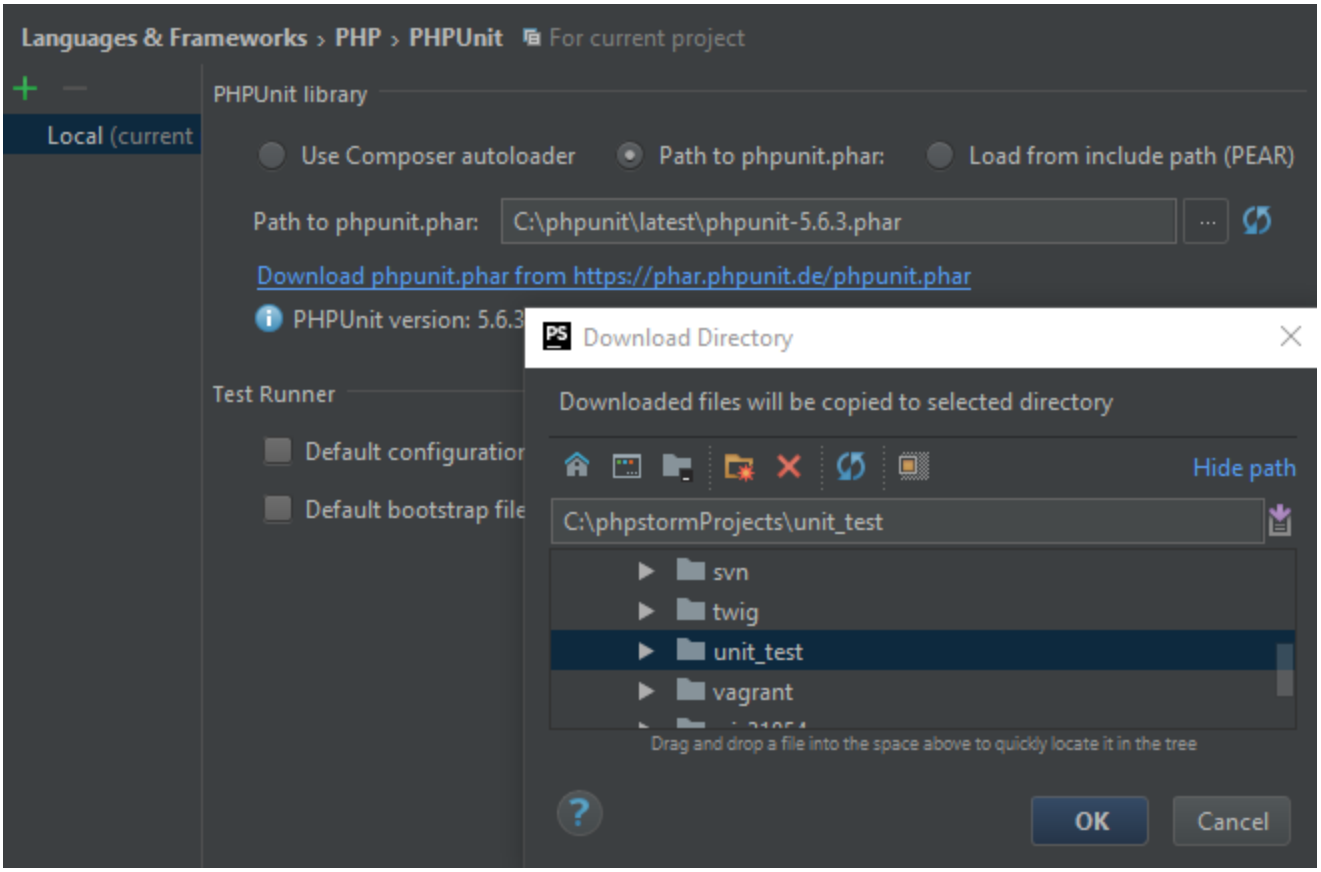
We can reference PHPUnit in several ways, depending on preferences or project standards.

- One approach is installing PHPUnit via Composer in PhpStorm.
- Another approach is downloading PHPUnit as a PHAR file and loading the PHAR file in our unit test code.
- And another approach is to install PHPUnit using PEAR but this is an obsolete option for PhpUnit < 5.

To get PHPUnit code completion PHAR file or Composer installation should either be in a project or explicitly included via Include Path.

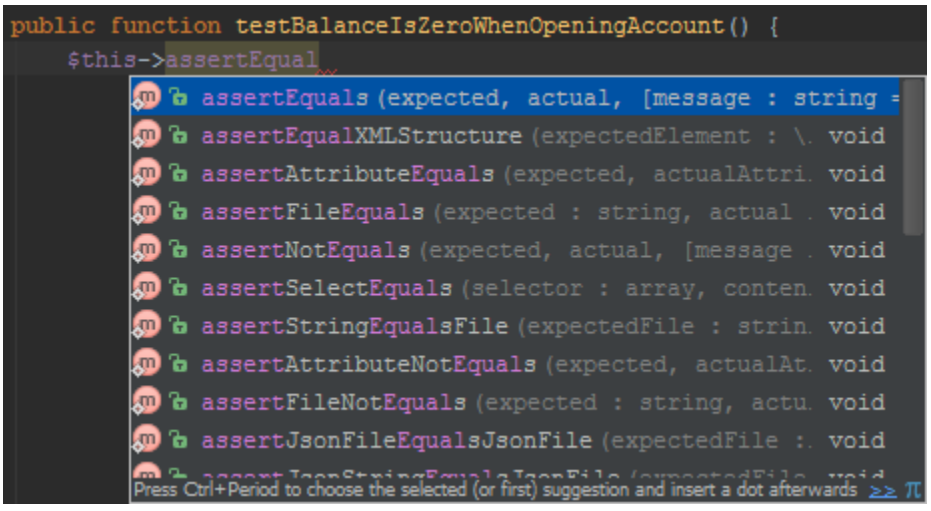
For this tutorial, let's go with PHPUnit support via PHAR file. From the Settings (Preferences on Mac) | Languages & Frameworks | PHP | PHPUnit, we can specify how we want to load PHPUnit. The options presented are the same options as described above (include path, Composer or PHAR). We don't have to grab the phpunit.phar file from www.phpunit.de ourselves: we can let PhpStorm download it for us.

Let's go ahead and download PHPUnit PHAR file into a root folder of our project.



Optionally, we can also specify the path to a phpunit.xml configuration file, or the path to a bootstrap file (to run arbitrary PHP code before unit tests run).

After closing the settings and giving IDE some time to index a new archive in a project, we now get full autocompletion support on all PHPUnit's classes and functions:



> ⚠ If you want to have an example class to test as well as a sample set of unit tests for it, checkout our Workshop project .

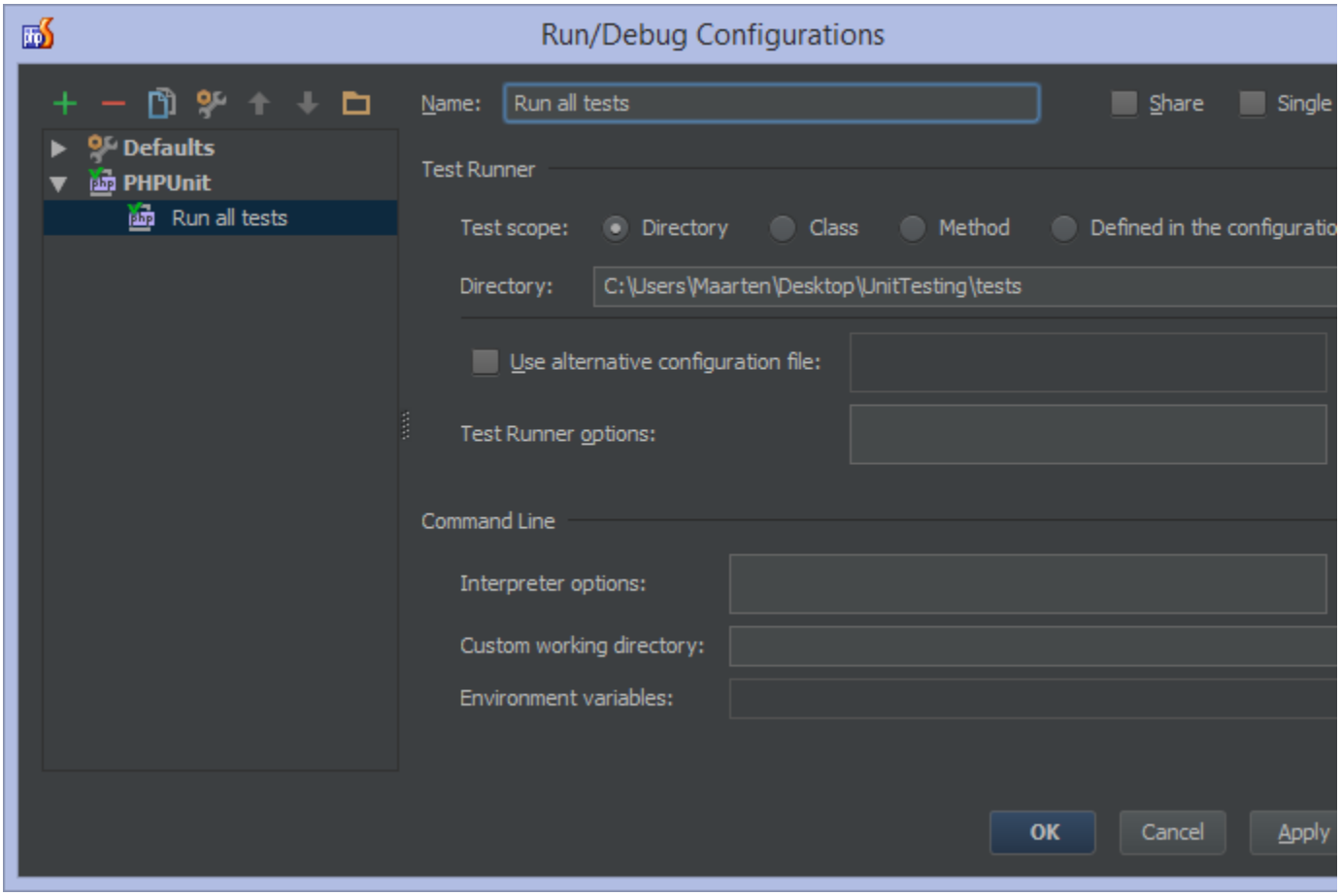
3. Running unit tests

The easiest way to run PHPUnit test in PhpStorm is to right click a test and select > Run 'test name'. A test class/separate method can be run by right clicking it directly in editor or by selecting a class file in a Project Tool window. Once run, PHPUnit run configuration will be automatically created: you can view it in Run menu > Edit configurations... . It should appear under

PHPUnit node and named after a test file we've run.

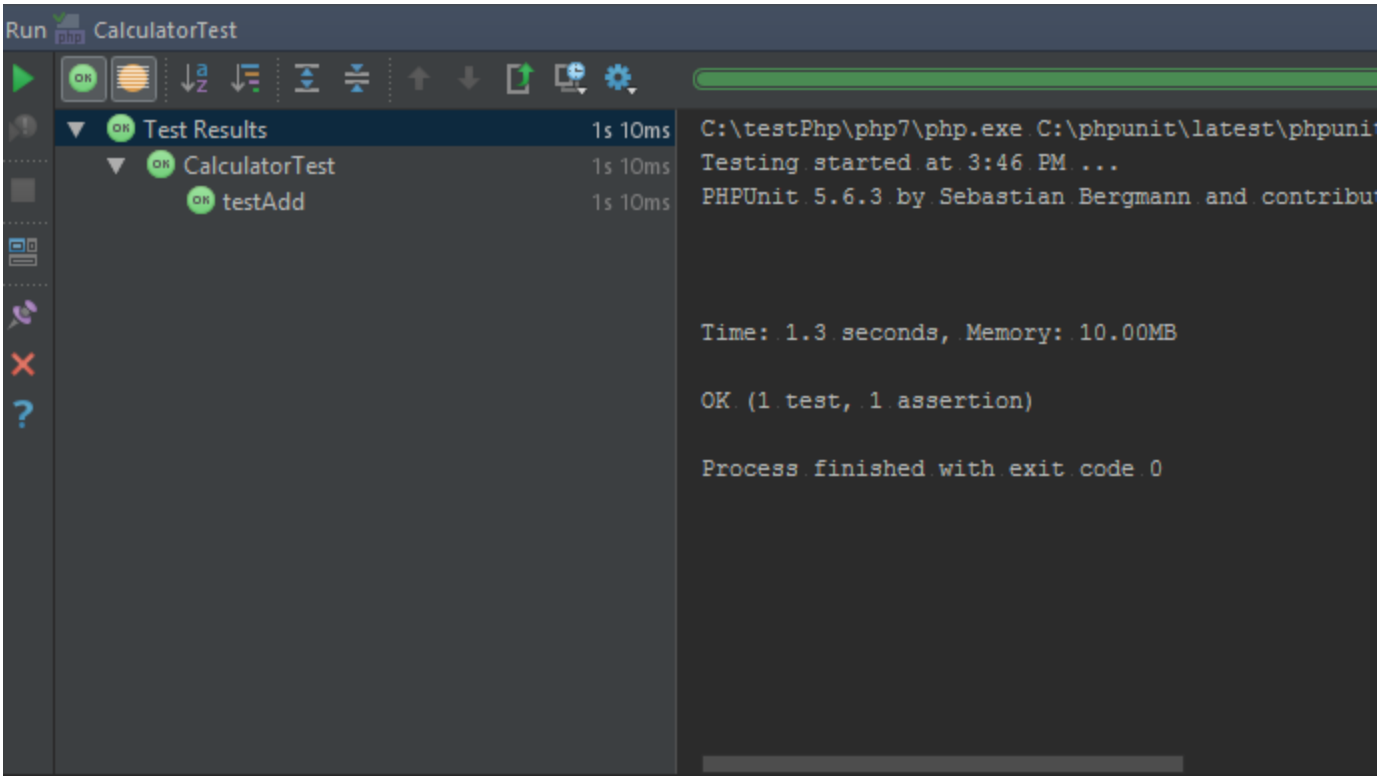
> We can right click a folder marked as Test Sources Root and select Run 'folder name' to run all the tests in selected folder.

We can also manually create a test run configuration but that it is often too cumbersome and should be reserved for the advanced PHPUnit test configurations. When we create a new PHPUnit run configuration (or edit existing one) we can provide details such as run configuration name, which tests should be run (all tests in a directory, a specific test class or one specific test). Adding test runner arguments or PHP interpreter options is also supported.



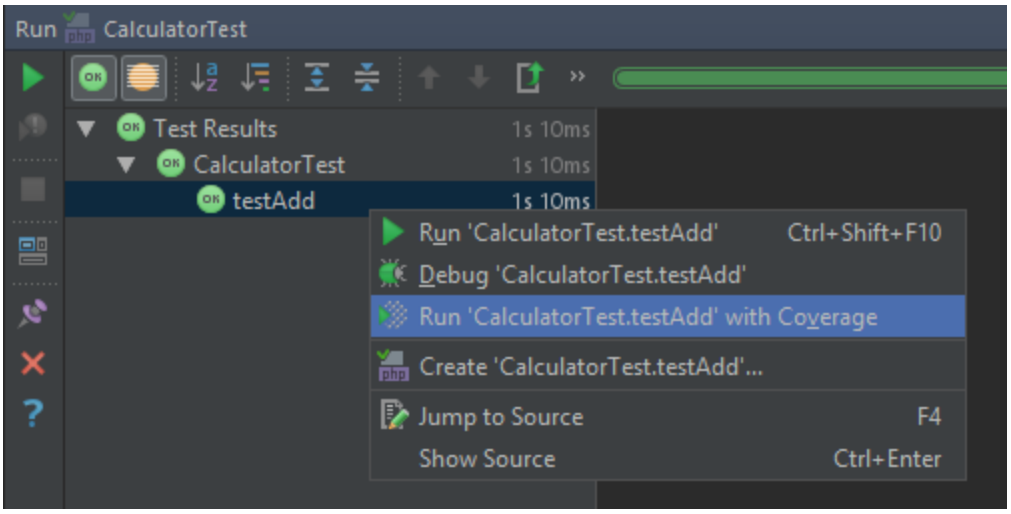
4. Reviewing test results

We are now ready to run (or debug) our unit tests! Let's do this: just right click a PHPUnit test and select "Run 'test name' ...". This will bring up a new tool window in which test results will be displayed.



The test results tool window is divided in 2 main areas: the left side allows us to filter tests or export results as well as drill down through all unit tests and see which ones succeeded and which ones failed. The right area shows us the raw PHPUnit output.

From the test results we can also use the context menu to perform several actions, such as running one specific test or navigating to the test source code.



Keep in mind that you can also debug unit tests, as well as use the techniques outlined in Profiling the Performance of PHP Applications to analyze the test run performance.

Finally, you can Rerun the tests. Rerunning failed tests only is also supported.

> See more info about Test Runner Tab at our online help page.

Tweet