

Tiki Unit Testing

This page describes how to write and run unit tests for Tiki, using PHPUnit. These tests are quite different from the kinds of tests that can be done with [TikiTest](#).

Unit tests, Integration tests and GUI level tests

In Tiki, we have three kinds of automated tests. This page refers to the first and second kind [unit tests](#) and [integration tests](#). For more information on GUI level tests, see [Tiki Testing with Selenium](#).

Cheat sheet / run the tests for the first time as a developer

First, follow the instructions at the top of file [lib/test/local.php.dist](#) [↗](#) to create the test database and configuration.

Then run the following to enter development mode. This will install and configure dependencies.

```
Enter Tiki Dev Mode
```

```
php console.php dev:configure
```

Finally, you should be able to either:

- **Run all tests**

```
php phpunit
```

- **Run a specific test method**

```
php phpunit --filter testNameOfTheTestMethodYouAreTryingToWriteOrFix
```

- **Run all the tests in a test class**

```
php phpunit --filter Tiki_Name_Of_TestClass
```

Then make sure all the dev dependencies are installed.

```
Install/update dev dependencies
```

```
php temp/composer.phar --ansi install -d vendor_bundled --no-progress --prefer-dist -n
```

Finally, you should be able to either:

- **Run all tests**

```
php lib/test/phpunit.php
```

- **Run a specific test method**

```
php lib/test/phpunit.php --filter testNameOfTheTestMethodYouAreTryingToWriteOrFix
```

- **Run all the tests in a test class**

```
php lib/test/phpunit.php --filter Tiki_Name_Of_TestClass
```

Unit tests

Unit Tests check the internals of certain classes, by exercising them in isolation from other classes and other infrastructure (eg: Apache, MySQL). This is what PHPUnit is especially good for. These tests run REALLY fast (the whole suite can be executed in less than a minute). Note however that those tests are only applicable to classes that are modular enough to be exercised in isolation from everything else. Generally speaking, that means the classes defined in lib/core.

Integration tests

Like [unit tests](#) integration tests check the internals of the system. The main difference is that they test one component in the context of the whole system, and, in particular, may need the Tiki DB in order to run. They are slower than Unit tests, but still faster than GUI level tests.

GUI level tests

These tests check the system by exercising it much in the same way that an end user might. This can be done either with [TikiTests](#) or [Selenium](#). This kind of test tends to be very slow and the whole suite may take several hours to run. This type of test is not covered by this page. For more details, see [Tiki Testing with Selenium](#).

Installing and configuring

Installing PHPUnit

PHPUnit is installed automatically via composer, when you run setup.sh. This doesn't seem to be true for trunk however. You need to `cd vendor_bundled` and run `php ../temp/composer.phar install` to install it on trunk.

Configuring environment variables

Some of the tests (those which restore the Tiki DB to start states) need to invoke `mysql` command. For this to work, you must make sure that the `mysql` command is part of your PATH.

Configuring a database for PHPUnit testing

Obviously we don't want our tests to mess up your actual production database. To prevent that, unit tests run on their own DB, which you have to create. You do this as follows.

First you have to create a new database. After you have to copy the file lib/test/local.php.dist to lib/test/local.php and edit it changing the default values to the values that match your environment.

The first time you run the test suite it will automatically populate the database. On subsequent runs it will update the database if needed.

Running existing tests

Once you have installed PHPUnit, you can run all the existing unit tests or subsets of them, as described in this section.

Ignoring pre-existing issues

In an ideal world, everyone would run the tests all the time, and would fix broken tests before doing a commit. But not everyone in our community does that, and therefore, it's quite common for trunk to have hundreds of broken tests that nobody knows how to fix.

When you run the tests, this makes it hard to tell if any of those failures or errors might be new, and have been introduced by a change you just made.

The `phpunit_with_baseline.php` script was designed to address this issue. Basically, it runs the tests and then compares the list of errors and failures to those of a baseline. It then only reports issues that are "new" and were not already present in that baseline.


To get more information on the usage of this script, do:

```
cd lib/test
php phpunit_with_baseline.php --help
```

Of particular interest is the `--phpunit-options` option. This is a string that will be passed on to phpunit. To know what the possibilities are for that option, do:

```
php vendor_bundled/vendor/phpunit/phpunit/phpunit --help
```


Note April 2016, Tiki 15:

The only way i  could get these tests to run was to use

```
php ../../bin/phpunit -c phpunit.xml
```

to run them all, or this for a subset

```
php ../../bin/phpunit -c phpunit.xml ./core/WikiParser
```

Also i have to create and update the test database using the normal installer - everything seems quite broken currently 

Setting up testing in PhpStorm

If you use PhpStorm you can use these instructions to set up PHPUnit to run on Tiki here: [Tiki Unit Testing with PhpStorm](#)

Setup the test database

You need to have a copy of `local.php` in the directory `lib/test`. You can create it by copying `lib/test/local.php.dist` and customizing it accordingly.

Run the tests

To run all the unit and integration tests (obsolete):

```
cd lib/test php phpunit_with_baseline
```

To run all the unit and integration tests on trunk (replace php with e.g. php71 if you use different php-cli versions):

```
php vendor_bundled/vendor/phpunit/phpunit/phpunit --colors=always
```

You may also exclude some tests like the GoogleTranslate tests, or the GUI level tests groups.

- GoogleTranslate tests don't work if you need to specify a web proxy, because they get some data from Google translation tool.
- [GUI level tests](#), require a selenium installation and a configuration file for selenium. You may not have this available.

So, if you don't want to run one of those groups, you just have to use the `--exclude-group` option, like this:

```
cd lib/test phpunit_with_baseline --phpunit-options "--exclude-group gui,GoogleTranslate"
```

Note that this assumes that a `phpunit @group` directive has been inserted in the source file of those tests.

You can also execute only tests whose names match a certain expression. For example:

1. Will only run tests whose name contains string `Importer`

```
phpunit_with_baseline --phpunit-options "--filter Importer"
```

Sometimes, PHPUnit fails and does not provide you with sufficient details to allow you to know exactly where the error occurred (for example, sometimes it doesn't even tell you in which `TestCase` the error happened). You can get more details by using the `--verbose` option. For example:

```
phpunit_with_baseline --phpunit-options "--verbose"
```

Writing new tests

Adding a new test function to an existing TestCase

If there is already a `TestCase` class (i.e. a class whose name ends with `Test` somewhere under `lib/test`) where your new test might fit in, then all you have to do is to create a new method of that class whose name starts with `"test"`. For example:

```
public function test_HelloWorld() { $this->fail("Forcing failure to see if this TestCase is actually loaded"); }
```

When you run the tests again, then you should see that `test_HelloWorld` was run and failed. Just change the content of that test so that it implements the actual tests.

Adding a new TestCase

If there isn't already a `TestCase` class where your new test might fit in, then you need to create a new

TestCase class. This is done slightly differently for the different types of tests.

Let's start with an example for a [unit tests](#). Say you want to create a class DummyUnitTest and want to put it under lib/core/Foo/Bar. All you need is to create a file lib/core/Foo/Bar/DummyUnitTest.php with the following content:

```
<?php /** @group unit */ class Foo_Bar_DummyUnitTest extends TikiTestCase { function setUp() {
parent::setUp(); /** Put your own setup code here. * Make sure you keep the above call to the parent
setUp() * in case the parent test case needs to do some * setup also. */ } function tearDown() { /** Put
your own setup code here. * Make sure you keep the call below to the parent tearDown() * to ensure
that the parent test case will teardown * anything it might have created. */ parent::tearDown(); } public
function test_HelloWorld() { $this->fail("Forcing failure to see if this TestCase is actually loaded"); } }
?>
```

Note:

- Foo_Bar_ prefix for the name of the class
 - This allows PHPUnit to automatically know that this class is located in a directory Foo/Bar located somewhere on the include path.
- The class name and the file ends with Test
- @group unit
 - This allows exclusion of unit tests using the --exclude-group option of PHPUnit

Creating a TestCase for an [integration test](#) is very similar, except that:

- You would put the file in a subdirectory of lib, instead of lib/core
- Set @group to 'integration' instead of 'unit'

For creation of [GUI level tests](#), details will follow.

Creating a new GUI level test

Details will follow.

Troubleshooting

PHPUnit hangs up

If you invoke PHPUnit and nothing at all happens, it's probably because your XAMP services are not started. Although you are running the tests from a command line, some of the tests may actually require some services like SQL to run.

Fatal error: Allowed memory size of NNN bytes exhausted

Running thousands of tests tends to consume much more memory than your average web PHP script, especially if you use a version of PHP that does not have garbage collection.

So you might get the above error. The way to address that is to increase your the memory_limit in your php.ini file.

Note that you have to make sure you use the proper php.ini file (there are often more than one on a given computer). To find out which php.ini file is used when you run PHPUnit, do this in a shell window:

```
php -i | grep ini
```

The file mentioned after "Loaded Configuration File =>" is the one you want. Just edit the file and increase the memory limit. If this is a development machine, then you can increase it all you want, but be careful if you are testing on a machine that is also used to deploy a Tiki site, and if the php.ini used by PHPUnit is also the one used by Apache, then don't increase it too much.

Tried to run an acceptance test without an initial database dump

If you get this error, it means you tried to run some [GUI level tests](#), but are not properly setup to run them. See [Tiki Testing with Selenium#Database_Issues](#) for details.

In order to run [GUI level tests](#), you must have snapshots of databases pre-configured for different tests. If you do not have those snapshots, the tests will not run, and all other tests will not run either.

A way around this problem is simply to exclude the GUI level tests using the --exclude-group option:

```
phpunit --exclude-group gui .
```

Failing tests for Multilingual_MachineTranslation_GoogleTranslateWrapperTest

This is probably due to the fact that you are using a web proxy. For some reason, the Google translation tool does not work from behind a proxy.

If you are running the tests behind a proxy, you can exclude those tests as follows:

```
phpunit --exclude-group GoogleTranslate .
```

WARNINGS reported as FAILURES

If your test executes code that raises a warning, this warning will get reported as a PHPUnit failure. That's because in phpunit.xml, we configure PHPUnit with convertWarningsToExceptions. That's because we want to be told about warnings and address them before they turn into actual bugs.

In some cases the code works just fine in spite of the warning, and getting rid of the warning would be too complicated to be worth it.

In those situations, you may want to address the issue by temporarily disabling E_WARNING just for that one bit of code, and just if you are running under tests. For example:

```
do { $old_error_reporting_level; if (defined('TIKI_IN_TEST')) { $old_error_reporting_level =  
error_reporting(E_ERROR | E_PARSE); } require_once("renderer_$function.php"); if  
(defined('TIKI_IN_TEST')) { error_reporting($old_error_reporting_level); } } while (false);
```

[alias](#)

[Unit Tests](#)

[Unit Testing](#)