Twin Cities New Years' Test Sprint
(A Drupal Event)

“New Years' Resolution: **Learn to write tests.**” Workshop

Led by Chach Sikes | http://chachaville.com | January 8, 2010

Twin Cities New Years' Test Sprint

Schedule

SATURDAY

Morning:

* 'Learn to write tests' workshop (10:30-11:30am)
* Pancake preparation and eating (10:30-12pm)

Afternoon:

12:00pm – 5pm: Test Sprint

* Orientation
* Little presentation on Test Writing for Drupal
* Form groups, work on test writing projects
* *4:30:* Wrap up and share what we've done with the group and Drupal community.

What will we do in this workshop?

1. Presentation: Testing in Drupal.

2. Downloads & Setup: Tools for doing tests.

3. Write or modify a test for a simple module.

Goal of the Workshop:
To get you started writing tests.

Getting set up to write tests is more a matter of learning where the tools are.

Writing good tests requires practice.

DISCLAIMER

I am not an expert.
I just started to learn test writing, too.

Testing in Drupal

Simply put, tests in Drupal are:

a way to program an invisible web browser to see if a module behaves as expected.

Technically...

Tests in Drupal:

- Use both an external library and module called 'SimpleTest'
- Are stored in a file called **.test*
- Are written with PHP's class syntax
- Have a few code standards of their own
- Use functions from SimpleTest and Drupal SimpleTest API
- Mainly use the special Drupal-specific functions
- Do not remember your Drupal settings, you must recreate them.
- Are run from a web-interface or from drush

- Can't do everything
- Require time and energy just like writing themes and modules
- Require a plan about what to test.
- Easy to get started, more difficult to understand what to test.

Why write tests?

- Required for Drupal core contributions and patches
- (Strongly recommended) for Drupal 7 contributed modules
- Automates checking to see if your module behaves as expected (i.e. less form-clicking)
- Requires you to think critically about how you write code, including breaking functions into smaller pieces
- Helps with collaborative code development

Kinds of tests

SimpleTest

Drupal's custom testing framework, built on Simpletest – PHP's Unit testing library – performs 'automated tests'

Unit testing

* Testing functions
* “A unit is the smallest testable part of an application” – usually a function (wikipedia)

Integration testing

* Testing modules, or parts of an application
* “...the phase in software testing in which individual software modules are combined and tested as a group” - could be a module. (wikipedia)

Browser testing

* testing that browsers behave as expected
* SimpleTest DrupalWebTestCase – has invisible browser
* JSUnit: javascript browser testing http://www.jsunit.net/
* Selenium: programming firefox to do tests

Test Writing Reality Check

- Tests take time to write
- Rewriting module may require rewriting tests
- Tests can be buggy
- Limited to capabilities of SimpleTest and Drupal's SimpleTest library (which doesn't test everything)
- Not likely to help with theme development

Demonstration:

Drupal's SimpleTest demonstration tutorial.

(a good example because it is a simple module, shows the test interface, and comes with useful commands for debugging your tests.)

http://drupal.org/node/395012

1. Set up your machine.

* Localhost (WAMP/MAMP/LAMP)
* Code editor
* Firefox

2. Download package for sprint.

github.com/chachasikes/TwinCitiesTestSprint

Collaboration tools:

* github

* minneapolis irc channel: #drupal-mpls

* http://etherpad.com/

* http://drupalbin.com/

Downloads

patched version of drupal 6

mymodule (from drupal.org demo)

mysettings (testing demo module)

mysettings/snippets/demo.php
(useful test php snippets)

command cheatsheet

drupal simpletest api documentation

Links

Drupal project page for SimpleTest
http://drupal.org/project/simpletest

Drupal SimpleTest tutorial
http://drupal.org/node/395012

A Drupal Module Developer's Guide to SimpleTest
http://www.lullabot.com/articles/drupal-module-developer-guide-simpletest

SimpleTest d6 Documentation
http://contribs.rocky-shore.net/simpletest/doxy/html/index.html

SimpleTest d6 Configuration
http://drupal.org/node/291740

Tips (buried in Simpletest documentation)
http://drupal.org/node/30011

Guidelines
http://drupal.org/node/325974

Create your first test for mysetting.module

mysetting module
/sites/all/modules

MySetting is a very simple module that allows a user to save a piece of data to the database

TASK 1: **Set up** a .test file

(There is a demonstration test included in the folder, but don't look at it yet.)

TASK 2: **Run the test** in the SimpleTest admin interface

TASK 3: Write code that will **test that data is stored to database**

Set up a .test file

1. create a file called mysetting/tests/mysetting.test

2. get the .test starter code from mysetting/snippets/empty.test

3. open mysetting/snippets/demo.test

Things to notice:

PHP Class Syntax
Setup
Naming

PHP Class Syntax

Examples from
Php Manual: **Classes and objects**

Example #2 Simple Class definition

```php
<?php
class SimpleClass
{
    // member declaration
    public $var = 'a default value';

    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

Example #5 Creating an instance

```php
<?php
$instance = new SimpleClass();

// This can also be done with a variable:
$className = 'Foo';
$instance = new $className(); // Foo()
?>
```

Example #6 Object Assignment

```php
<?php
$assigned   =  $instance;
$reference  =& $instance;

$instance->var = '$assigned will have this value';

$instance = null; // $instance and $reference become null

var_dump($instance);
var_dump($reference);
var_dump($assigned);
?>
```

The above example will output:

```
NULL
NULL
object(SimpleClass)#1 (1) {
   ["var"]=>
     string(30) "$assigned will have this value"
}
```

Example #7 Simple Class Inheritance

```php
<?php
class ExtendClass extends SimpleClass
{
    // Redefine the parent method
    function displayVar()
    {
        echo "Extending class\n";
        parent::displayVar();
    }
}

$extended = new ExtendClass();
$extended->displayVar();
?>
```

The above example will output:

```
Extending class
a default value
```

```php
/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // We can redeclare the public and protected metho
d, but not private
    protected $protected = 'Protected2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // Works
echo $obj2->private; // Undefined
echo $obj2->protected; // Fatal Error
$obj2-
>printHello(); // Shows Public, Protected2, Undefined

?>
```

Scope Resolution Operator (::)

Example #3 Calling a parent's method

```php
<?php
class MyClass
{
    protected function myFunc() {
        echo "MyClass::myFunc()\n";
    }
}

class OtherClass extends MyClass
{
    // Override parent's definition
    public function myFunc()
    {
        // But still call the parent function
        parent::myFunc();
        echo "OtherClass::myFunc()\n";
    }
}

$class = new OtherClass();
$class->myFunc();
?>
```

Scope operator

Classes + extends

Calling functions from the main class

Init()

Each function is called in sequence

Look at Simple Test Documentation

Look at Simple Test Documentation