

The PHPSpec 0.2.0dev API

I had a request to make a very quick summary of the [PHPSpec](http://dev.phpspec.org) API for the current development snapshots over on <http://dev.phpspec.org>. We're lacking documentation for right now, so here's a brief overview.

First of all, since this is a relatively short-life development period, there are a few basic assumptions in place for now.

1. One spec class per file
2. The file name reflects the classname
3. Spec classnames start with "Describe"

A current valid spec class would be similar to

```
class DescribeNewLoggerUsingFileStorage extends PHPSpec_Context
{
    public function itShouldCreateCreateNewFileIfNoneExists()
    {
        $this->pending();
    }
    public function itShouldUseAnExistingFileIfOneExistsWithoutTruncatingIt()
    {
        $this->pending();
    }
}
```

This based on a plain text spec such as:

New logger using file storage:

- should create new file if none exists
- should use an existing file if one exists without truncating it

So the filename in this case would be DescribeNewLoggerUsingFileStorage.php. In reality there should be a less restrictive naming scheme but for now it narrows the possible variations while development is progressing.

The PHPSpec_Context class is a bit like your typical xUnit TestCase class. It's the ultimate parent for all specs. It's called a Context because each class should describe a class or system of classes for a "Given" or context, i.e. the condition or environment of the class being tested. In the case above, the context is that we've just instantiated a Logger which writes messages to a file.

PHPSpec_Context::pending() simple marks an example as awaiting completion.

We can flesh out the spec's examples (note: a spec is a collection of executable examples) as follows.

```
class DescribeANewLoggerUsingFileStorage
{
    public function itShouldCreateNewFileIfNoneExists()
```

```

{
    $file = $this->getTmpFileName();
    $logger = new Logger( $file );
    $this->spec(file_exists($file))->should->beTrue();
}
public function itShouldUseAnExistingFileIfOneExistsWithoutTruncatingIt()
{
    $file = $this->getTmpFileName();
    file_put_contents($file, 'Hello' . "\n");
    $logger = new Logger( $file );
    $this->spec(file_get_contents($file))->should->be('Hello' . "\n");
}
public function after()
{
    unlink($this->getTmpFileName());
}
public function getTmpFileName()
{
    return sys_get_temp_dir() . DIRECTORY_SEPARATOR . 'logger_tmp_file';
}
}

```

You can execute a spec by using the command line script "phpspec" available also for Windows as a batch file.

```
phpspec DescribeANewLoggerUsingFileStorage
```

You can also simple execute all specs within a directory tree using.

```
phpspec -r
```

Specs can also be continually re-run using the -a flag.

Back to the full spec example, PHPSpec_Context::spec() method returns an implementation of a Domain Specific Language (DSL) which basically means it's a specific language designed for Behaviour-Driven Development. This contrasts to an xUnit API based on assertions. The DSL is intended as being more readable, and more intuitive to write examples with.

A very basic call above uses the "be" Matcher, and the "should" Expectation. In PHPSpec, you can state whether any Matcher "should" or "shouldNot" pass. The Matchers presently available (by no means a complete list sufficient for general usage quite yet):

```

be()
equal()
beEqualTo()
beAnInstanceOf()
beGreaterThan()
beLessThan()
beGreaterThanOrEqualTo()
beLessThanOrEqualTo()
beEmpty()

```

beTrue()
beFalse()


Scheduled for the next day or two:

beBetween()
beNull()
beOfType()
beIdenticalTo()
beSet()
match() // PCRE Regular Expressions
raise() // Exceptions
trigger() // Errors

In addition, I recently implemented "Predicate" matching. Predicates are basically methods starting with "is" or "has", e.g. hasMessages(), isPrepared(), hasPlayer(), present in your implementation classes. Using PHPSpec's predicate matcher you can write an example such as:


```
$this->spec($someObject)->should->haveMessages();
```

This calls \$someObject->hasMessages(), and compares the result to the boolean TRUE to ascertain a match. Of course the should/shouldNot expectations work here also to interpret whether a matcher result should pass or fail.

Output from [PHPSpec](#) is still a bit unintelligent. I have updated it to follow the traditional xUnit output (same style as PHPUnit for example) but data such as error line numbers, and backtraces, and string comparison results are not yet possible. Obviously a big priority before any public release could be finalised 

We're homing in on an initial public release but I'll hold off until minimal documentation is in place. In the meantime, you can download regularly updated snapshots from <http://dev.phpspec.org>. Note that the tar.gz file is a typical archive, and the .tgz file is a PEAR installable package. The second is much preferred since it will install the necessary script files.

Posted by Pádraic Brady in PHP General, PHP Security at 18:47

Looks very promising 

Would be nice if I could write

```
$this->spec($someValue)->should->satisfy(new MyCustomMatcher($expectedValue));
```

Anonymous on Nov 7 2007, 02:37