Tests\Testing\Template

The ***Tests\Testing\Template*** class is a sample test class which demonstrates the minimum requirements for extending the ***Library\Testing\Base*** class.

Design criteria and considerations

The ***Tests\Testing\Template*** class

1. extends the ***Library\Testing\Base*** class;
2. uses the ***Tests\Testing*** namespace.

Class Methods

The ***Tests\Testing\Template*** class contains the following methods:

Method	Description
__construct	Class constructor.
__destruct	Class destructor.
assertionTests	Execute the assertion tests.
a_trueTest	Sample true test assertion.
a_falseTest	Sample false test assertion.

__construct

Class constructor.

Algorithm

1. Call the ***__construct*** class method of the **parent** class;
2. pass the constants **1**, **0**, **0**, and an array containing '**Library\Testing\Base**' and '**assertCallback**' to the **assertSetup** class method;
3. exit.

Implementation

```
/**
 * __construct
 *
 * Class constructor
 */
public function __construct()
{
	parent::__construct();
	$this->assertSetup(1, 0, 0, array('Library\Testing\Base', 'assertCallback'));
}
```

Narrative

The first step is to call the ***Library\Testing\Base*** constructor using the PHP **parent** scope resolution operator, used to access overriden super-class methods.

The assertSetup method, defined in ***Library\Testing\Base***, is called to set the PHP **assert** options. Refer to the ***Library\Testing\Base*** class documentation for more details.

__destruct

The class destructor.

Algorithm

1. Call the **__destruct** class method of the **parent** class;
2. exit.

Implementation

```
/**
 * __destruct
 *
 * Class destructor.
 */
public function __destruct()
{
	parent::__destruct();
}
```

Narrative

The ***Library\Testing\Base*** destructor is called using the PHP **parent** scope resolution operator, used to access overriden super-class methods.

assertionTests

Performs the assertion test steps.

Algorithm

1. Accept ***logger*** as the optional name of the logger to use;
2. accept ***format*** as the optional log output format;
3. assign the ***eolSequence*** property of the ***properties*** class property to the ***eolSequence*** class property;
4. if ***logger*** is null, goto step **6**;
5. pass ***logger*** and ***format*** to the **assertStartLogger** class method;
6. pass **false** to the **assertFailures** class method;
7. call the **a_trueTest** class method;
8. call the **a_falseTest** class method;
9. exit.

Implementation

```
/**
 * assertionTests
 *
 * run the current assertion test steps
 * @param string $logger = (optional) name of the logger to use, null for none
 * @param string $format = (optional) log output format
 */
public function assertionTests($logger=null, $format=null)
{
    $this->eolSequence = $this->properties->eolSequence;

    if ($logger !== null)
    {
        $this->assertStartLogger($logger, $format);
    }

    $this->assertFailures(false);

    $this->a_trueTest();
    $this->a_falseTest();
}
```

Narrative

The ***assertionTests method*** expects two optional parameters:

Parameter	Description
$logger	Name of the logger to use, null to use the default.
$format	Log output format.

The ***eolSequence*** class property is set to the value set by ***Library\Testing\Runner*** class in the ***eolSequence*** property of the ***properties*** class property.

If the ***$logger*** parameter is not **null**, the ***assertStartLogger*** method is passed the

values in the ***$logger*** and ***$format*** parameters to start the test logger.

The ***assertFailures*** method is passed a value of **false** to disable the reporting of assert failures.

The remainder of the ***assertionTests*** method is dedicated to the implementation of the steps required to completely exercise the methods and functionality of the class under test.

a_trueTest

A sample of calling the ***asertTrue*** method with a true assertion.

Algorithm

1. Create a command string containing '**(1 + 1) == 2;**';
2. assign the command string to ***assertion***;
3. pass ***assertion***, and an assertion message to be printed, to the **assertTrue** class method;
4. if the result is **true**, goto step **8;**
5. pass the string '**(1 + 1) != 2**' to the **assertLogMessaage** class method;
6. exit the **phpTest** application with a result of **1**;
7. pass the string '**Test result is TRUE**' to **assertLogMessage**;
8. exit.

Implementation

```
/**
 * a_trueTest
 *
 * A sample of calling the asertTrue method with a true statement
 */
public function a_trueTest()
{
	$this->labelBlock('True Test', 40);

	$assertion = '(1 + 1) == 2;';
	if (! $this->assertTrue($assertion, sprintf('Asserting: %s', $assertion)))
	{
		$this->assertLogMessage('(1 + 1) != 2');
		exit(1);
	}
	$this->assertLogMessage('Test result is TRUE');
}
```

Narrative

The ***labelBlock*** method is passed a block title ('**True Test**') and a block size (**40 characters**) to be output to the current logging device(s).

A true assertion of '**1 + 1 == 2**' is set in the ***$assertion*** method variable and passed to the ***assertTrue*** method, along with a message to be output to the current logging device(s).

If the value returned from the ***assertTrue*** method is false,

the ***assertLogMessage*** class method is called to output a message **('1 + 1 != 2)**';

the test program, **phpTest**, is exited.

The ***assertLogMessage*** class method is called to output the message '**Test result is TRUE**'.

a_falseTest

A sample of calling the ***asertFalse*** method with a false assertion.

Algorithm

1. Create a command string containing '**(1 + 1) == 3;**';
2. assign the command string to ***assertion***;
3. pass ***assertion***, and an assertion message to be printed, to the **assertFalse** class method;
4. if the result is **true**, goto step **7**;
5. pass the string '**(1 + 1) == 3**' to the **assertLogMessaage** class method;
6. exit the **phpTest** application with a result of **1**;
7. pass the string '**Test result is FALSE**' to **assertLogMessage**;
8. exit.

Implementation

```
/**
 * a_falseTest
 *
 * A sample of calling the asertFalse method with a false assertion
 */
public function a_falseTest()
{
	$this->labelBlock('False Test', 40);

	$assertion = '(1 + 1) == 3;';
	if (! $this->assertFalse($assertion, sprintf('Asserting: %s', $assertion)))
	{
		$this->assertLogMessage('(1 + 1) == 3');
		exit(1);
	}

	$this->assertLogMessage('Test result is FALSE');
}
```

The ***labelBlock*** method is passed a block title ('**False Test**') and a block size (**40 characters**) to be output to the current logging device(s).

A true assertion of '**1 + 1 == 3**' is set in the ***$assertion*** method variable and passed to the ***assertFalse*** method, along with a message to be output to the current logging device(s).

If the value returned from the ***assertFalse*** method is false,

the ***assertLogMessage*** class method is called to output a message **('1 + 1 == 3)**';

the test program, **phpTest**, is exited.

The ***assertLogMessage*** class method is called to output the message '**Test result is FALSE**'.

Source Listing

```
<?php
namespace Tests\Testing;

/*
 *     Testing\Template is copyright © 2012, 2013. EarthWalk Software.
 *     Licensed under the Academic Free License version 3.0
 *     Refer to the file named License.txt provided with the source,
 *           or from http://opensource.org/licenses/academic.php
 */
/**
 * Template
 *
 * Testing\Template sample test.
 * @author Jay Wheeler
 * @version 1.0
 * @copyright © 2012, 2013 EarthWalk Software.
 * @license Licensed under the Academic Free License version 3.0.
 * @package Tests
 * @subpackage Testing
 */
class Template extends \Library\Testing\Base
{
	/**
	 * __construct
	 *
	 * Class constructor - pass parameters on to parent class
	 */
	public function __construct()
	{
		parent::__construct();
		$this->assertSetup(1, 0, 0, array('Library\Testing\Base', 'assertCallback'));
	}

	/**
	 * __destruct
	 *
	 * Class destructor.
	 */
	public function __destruct()
	{
		parent::__destruct();
	}

	/**
	  * assertionTests
	  *
	  * run the current assertion test steps
	  * @param string $logger = (optional) name of the logger to use, null for none
	  * @param string $format = (optional) log output format
	  */
	 public function assertionTests($logger=null, $format=null)
	 {
		$this->eolSequence = $this->properties->eolSequence;

		 if ($logger !== null)
		 {
			$this->assertStartLogger($logger, $format);
		 }
```

```
            $this->assertFailures(false);

            $this->a_trueTest();
            $this->a_falseTest();
      }

      /**
       * a_trueTest
       *
       * A sample of calling the asertTrue method with a true assertion
       */
      public function a_trueTest()
      {
            $this->labelBlock('True Test', 40);

            $assertion = '(1 + 1) == 2;';
            if (! $this->assertTrue($assertion, sprintf('Asserting: %s', $assertion)))
            {
                  $this->assertLogMessage('(1 + 1) != 2');
                  exit(1);
            }

            $this->assertLogMessage('Test result is TRUE');
      }

      /**
       * a_falseTest
       *
       * A sample of calling the assertFalse method with a false assertion
       */
      public function a_falseTest()
      {
            $this->labelBlock('False Test', 40);

            $assertion = '(1 + 1) == 3;';
            if (! $this->assertFalse($assertion, sprintf('Asserting: %s', $assertion)))
            {
                  $this->assertLogMessage('(1 + 1) == 3');
                  exit(1);
            }

            $this->assertLogMessage('Test result is FALSE');
      }
}
```