Testing\Base

The ***Testing\Base*** class is a base class for creating assertion class tests. The ***Testing\Base*** class is designed to be extended by the class tests.

Design criteria and considerations

The ***Testing\Base*** class

1. is designed as a base class, providing the functionality required for writing class assertion tests;
2. uses the ***Library\Testing*** namespace;
3. uses the ***Library\CliParameters*** class for passing run-time options to the test class;
4. uses the ***Library\Exception\Descriptor*** class for recording test class exceptions;
5. uses the ***Library\Log*** and ***Log\Format*** classes for logging operations;
6. uses the ***Library\PrintU\FormatArray*** class to format array output;
7. uses the ***Testing\Setup*** class for setup information;
8. uses the ***Library\Testing\Exception*** class to throw exceptions;
9. uses the ***Library\Autoload*** class to load dependent classes;
10. uses the ***Library\Properties*** class as a container class;
11. uses the ***Library\Testing\Setup*** class to access ***phpTest*** properties.

Class Properties

The ***Testing\Base*** class contains the following properties (variables):

Method	Description
$properties	***Properties*** class instance
$exceptionCaught	Exception caught in last assertion if true.
$exception	***Exception\Descriptor*** class instance.
$exceptionCallback	User-defined exception callback method.
$verbose	Allows output of messages when not 0.
$reportFailures	True to report assert failures, even if ***$verbose*** = 0.
$assertLogger	Name of the assert logger internal name.
$loggerProperties	A ***Library\Properties*** object for by-test logging properties
$errorsOnly	Non-zero to output error messages only on assert failure.
$testNumber	Current sub-test number.
$testName	Name of the test being run.
$assert	An array containing information about the last false assertion processed
$labelBlock	True to allow output of label blocks, when a label block is requested.
$cliParameters	A ***Library\CliParameters*** object for by-test run-time (CLI) variables.
$eolSequence	Current end-of-line sequence.

The ***Testing\Base*** class contains the following static property (variable):

Method	Description
$me	Contains a copy of the Testing\Base class instance object for static class tests.

Class Methods

The ***Testing\Base*** class contains the following methods:

Method	Description
__construct	Class constructor
__destruct	Class destructor
assertSetup	Set assert options
assertTest	Asserts the assertion and returns the result.
assertException	Assert the assertion and return the result conditioned by ***$expected*** and the ***$exceptionCaught*** class property.
assertTrue	Perform an assertion that the test is true.
assertFalse	Perform an assertion that the test is false.
assertExceptionTrue	Perform an assert that the test is **true.** Returns **true** if the assertion is **true**, otherwise returns **false**.
assertExceptionFalse	Perform an assert that the test is **false.** Returns **true** if the assertion is **false**, otherwise returns **false**.
getException	Return the current exception object.
getExceptionCode	Return the code associated with the last exception.
getExceptionMessage	Return the message associated with the last exception.
getExceptionClass	Return the class name of the last exception.
exceptionCaught	Return the ***$exceptionCaught*** class property.
labelBlock	Output a separator block containing a label.
labelBlockFlag	Get/set the label block flag.
assertCallback	Assert test callback function.
properties	Get/set the properties object.
verbose	Get/set the verbose property.
errorsOnly	Display only errors flag, used in **assertTest** to limit output.
testNumber	Get/set the sub-test number.
testName	Get/set the test name.

Method	Description
assertEventArray	Get/set the event array.
assertLogMessage	Output a formatted message to the current output device(s).
assertFailures	Get/set the ***$reportFailures*** flag.
assertExceptionDescriptor	Get a copy of the exception descriptor.
getEldestParent	Get the top parent in the hierarchy.
assertExceptionCallback	Get/set the exception callback function.
assertStartLogger	Create a test-relative log file and start the logger.
assertStopLogger	Stop the named logger.

__construct

Class constructor.

Algorithm

1. Set the ***properties***, ***loggerProperties***, ***exception***, ***exceptionCallback***, and ***eolSequence*** class properties to **null**;
2. set the ***exceptionCaught*** to **false**;
3. set the ***reportFailures*** and ***labelBlock*** class properties to **true**;
4. set the ***assertEvent*** class property to an empty array;
5. set the ***verbose*** and ***testNumber*** class properties to **0**;
6. set the ***errorsOnly*** class property to 1;
7. set the ***testName*** class property to a string containing '**<unknown>**';
8. set the static class property ***me*** to a reference to this class;
9. pass the string '**\Library\CliParameters**' to the **loadClass** method of the **\Library\Autoload** class;
10. exit.

Implementation

```
/**
 * __construct
 *
 * Class constructor
 */
public function __construct()
{
	$this->properties = null;
	$this->loggerProperties = null;

	$this->exception = null;
	$this->exceptionCaught = false;
	$this->exceptionCallback = null;

	$this->assertEvent = array();

	$this->verbose = 0;
	$this->errorsOnly = 1;
	$this->testNumber = 0;
	$this->testName = '<unknown>';
	$this->eolSequence = null;

	$this->reportFailures = true;
	$this->labelBlock = true;

	\Library\Autoload::loadClass('\Library\CliParameters');

	self::$me = $this;
}
```

Narrative

All class properties (variables) are set to default values.

__destruct

Class destructor.

Algorithm

1. exit.

Implementation

```
/**
 * __destruct
 *
 * Destroy the current class instance
 */
public function __destruct()
{
}
```

Narrative

The class **destructor** method s included for completeness. Although it currently is an empty class method, it may be populated in future versions. Therefore, all class that extend this class ***should*** implement a class **destructor** method and call this **destructor** method on exit.

assertSetup

Setup assert options.

Algorithm

1. Accept ***active*** as an (optional) integer indicating the desired setting of the PHP **ASSERT_ACTIVE** assert option;
2. accept ***warning*** as an (optional) integer indicating the desired setting of the PHP **ASSERT_WARNING** assert option;
3. accept ***bail*** as an (optional) integer indicating the desired setting of the PHP **ASSERT_BAIL** assert option;
4. accept ***callback*** as an array containing the desired setting of the PHP **ASSERT_CALLBACK** assert option;
5. pass the PHP **ASSERT_ACTIVE** constant and ***active*** to the PHP **assert_options** function;
6. pass the PHP **ASSERT_WARNING** constant and ***warning*** to the PHP **assert_options** function;
7. pass the PHP **ASSERT_BAIL** constant and ***bail*** to the PHP **assert_options** function;
8. pass the PHP **ASSERT_CALLBACK** constant and ***callback*** to the PHP **assert_options** function;
9. exit.

Implementation

```
/**
 * assertSetup
 *
 * Set assert options
 * @param integer $active = (optional) active flag: 0 ==> inactive, 1 ==> active.
 *                                  (Default: 1)
 * @param integer $warning  = (optional) warning flag: 0 ==> silent, 1 ==> issue
 *                                  warning on assert failure. (Default: 0)
 * @param integer $bail     = (optional) bail-out flag: 0 ==> keep processing, 1 ==>
 *                                  abort processing. (Default: 0)
 * @param string  $callback = (optional) callback function name.
 *                                  (Default = \Library\Testing\Base::assertCallback()).
 */
protected function assertSetup($active=1, $warning=0, $bail=0,
                               $callback=array('\Library\Testing\Base',
                                               'assertCallback'))
{
      assert_options(ASSERT_ACTIVE,    $active);
      assert_options(ASSERT_WARNING,   $warning);
      assert_options(ASSERT_BAIL,      $bail);
      assert_options(ASSERT_CALLBACK,  $callback);
}
```

Narrative

The ***assertSetup*** method expects four optional parameters:

Parameter	Description
$active	Active flag: 0 = inactive, 1 = active.
$warning	Warning flag: 0 = silent, 1 = issue warning.
$bail	Bail-out flag: 1 = abort processing, 0 = don't.
$callback	Callback function name.

The following PHP ***assert_options*** flags are set to the value in the associated parameter variable:

Parameter	Assert Flag
$active	**ASSERT_ACTIVE**
$warning	**ASSERT_WARNING**
$bail	**ASSERT_BAIL**
$callback	**ASSERT_CALLBACK**

assertTest

Assert the assertion and return the result conditioned by ***$expected***.

Algorithm

1. Accept **assertion** as a string containing the assertion to test;
2. accept **message** as a string to print;
3. accept ***expected*** as a boolean containing the expected result;
4. increment the ***testNumber*** class property;
5. if the ***verbose*** class property is greater than **0** and ***message*** is not **null**, pass ***message*** to the **assertLogMessage** class method;
6. pass ***assertion*** to the PHP **assert** function;
7. if the result is **true**, goto step **13**;
8. if ***expected*** is **true**, goto step **11**;
9. if the ***verbose*** class property is greater than **0** and ***errorsOnly*** is **false**, pass the string '**Assertion is false**' to the **assertLogMessage** class method;
10. exit with a **true** result;
11. pass a string containing '**Assertion is FALSE but expected TRUE**' to the **assertLogMessage** class method;
12. exit with a **false** value;
13. if expected is not **true**, goto step **16**;
14. if the ***verbose*** class property is greater than **0** and ***errorsOnly*** is **false**, pass the string '**Assertion is false**' to the **assertLogMessage** class method;
15. exit with a **true** result;
16. pass a string containing '**Assertion is TRUE but expected FALSE**' to the **assertLogMessage** class method;
17. goto step **12**.

Implementation

```
/**
 * assertTest
 *
 * Assert the assertion and return the result conditioned by $expected
 * @param string $assertion = assertion to test
 * @param string $message = (optional) string to print if $verbose is true,
 *                                     null for no message
 * @param boolean $expected = (optional) expected result (boolean true or false)
 * @return boolean true = result was as expected, false = it was not as expected
 */
protected function assertTest($assertion, $message=null, $expected=true)
{
	++$this->testNumber;
	if ($this->verbose && ($message !== null))
	{
		$this->assertLogMessage($message);
	}

	if (! assert($assertion))
	{
		if ($expected == false)
		{
			if ($this->verbose && (! $this->errorsOnly))
```

```
                {
                        $this->assertLogMessage("Assertion is false");
                }

                return true;
        }

        $this->assertLogMessage
                ("***  Assertion is FALSE but expected TRUE ***");
        return false;
    }

    if ($expected == true)
    {
        if ($this->verbose && (! $this->errorsOnly))
        {
                $this->assertLogMessage("Assertion is true");
        }

        return true;
    }

    $this->assertLogMessage("***  Assertion is TRUE but expected FALSE  ***");
    return false;
}
```

Narrative

The ***assertTest*** method expects a single, mandatory parameter:

Parameter	Description
$assertion	The test assertion to be executed.

and two optional parameters:

Parameter	Description
$message	String to print if $verbose is true.
$expected	Expected result: true or false.

The ***assertTest*** method starts by incrementing the current sub-test number in the ***$testNumber*** class property.

If the ***$verbose*** class property is non-zero and the ***$message*** parameter is not null, the string in the ***$message*** parameter is passed to the ***assertLogMessage*** method for output to the testing log(s).

The PHP ***assert function*** is called and passed the ***$assertion*** parameter string to be tested. If a **true** value is returned,

- if the ***$expected*** parameter is **true**,
  - if ***$verbose*** class property is not 0 and the ***$errorsOnly*** class property is false, the ***assertLogMessage*** method is called to output "***Assertion is true***";

    - a **true** result is returned.
- Otherwise, the ***assertLogMessage*** method is called to output **"*** *Assertion is TRUE but expected FALSE* ***"** and a **false** result is returned.

If a **false** value is returned,

- if the ***$expected*** parameter is **false**,
    - if ***$verbose*** class property is not 0 and the ***$errorsOnly*** class property is false, the ***assertLogMessage*** method is called to output **"*Assertion is false*"**;
    - a **true** result is returned.
- Otherwise, the ***assertLogMessage*** method is called to output **"*** *Assertion is FALSE but expected TRUE* ***"** and a **false** result is returned.

assertException

Assert the assertion and return the result conditioned by ***$expected*** and the ***$exceptionCaught*** class property.

Algorithm

1. Accept ***assertion*** as a string containing the assertion to test;
2. accept ***message*** as a string containing the string to print;
3. accept ***expected*** as a boolean containing the expected result;
4. create a PHP **try** block;
5. assign false to the ***exceptionCaught*** class property;
6. pass ***assertion***, ***message*** and ***expected*** to the **assertTest** class method and return the result;
7. end the PHP **try** block;
8. create a PHP **catch** block;
9. catch a thrown exception of the PHP **Exception** class and assign the result to ***exception***;
10. if the **exceptionCallback** function is null, goto step **13**;
11. pass ***exception*** to the **exceptionCallback** function;
12. goto step 22;
13. set the ***exceptionCaught*** class property to **true**;
14. create an array with '**testname**' as the key and the ***testName*** class property as the field;
15. add '**testnumber**' as a new key in the array with the ***testNumber*** class property as the field;
16. using the PHP **new** command, pass ***exception*** and the array to the **Library\Exception\Descriptor** class to create a new class instance;
17. assign the new class instance to the ***exception*** class property;
18. if the ***reportFailures*** class property is true, goto step **21**;
19. concatenate the ***testname*** property of the ***exception*** class property, the ***testnumber*** property of the ***exception*** class property, and the ***exception*** class property into a string;
20. pass the string to the **assertLogMessage** class method;
21. end the PHP **catch** block;
22. exit with a **false** value.

Implementation

```
/**
 * assertException
 *
 * Assert the assertion and return the result conditioned by
 *     $expected and $exceptionCaught
 * @param string $assertion = assertion to test
 * @param string $message = (optional) string to print if $verbose is true
 * @param boolean $expected = (optional) expected result
 * @return boolean true = result was as expected, false = not
 */
protected function assertException($assertion, $message=null, $expected=true)
{
	try
	{
		$this->exceptionCaught = false;
		return $this->assertTest($assertion, $message, $expected);
	}
	catch(\Exception $exception)
	{
		if ($this->exceptionCallback)
		{
			$this->{$this->exceptionCallback}($exception);
		}
		else
		{
			$this->exceptionCaught = true;
			$this->exception = new \Library\Exception\Descriptor($exception,
					array('testname'    => $this->testName,
					      'testnumber' => $this->testNumber));
			if ($this->reportFailures)
			{
				$this->assertLogMessage(sprintf("%s Test #%u - %s",
					$this->exception->testname,
					$this->exception->testnumber,
					$this->exception));
			}
		}
	}

	return false;
}
```

Narrative

The ***assertException*** method expects a single, mandatory parameter:

Parameter	Description
$assertion	The test assertion to be executed.

and two optional parameters:

Parameter	Description
$message	String to print if $verbose is true.
$expected	Expected result: true or false.

The ***assertException*** method resets the ***$exceptionCaught*** class property to false and calls the ***assertTest*** method to process the ***$assertion*** parameter, after setting up a try-catch block to catch any exception thrown.

If an exception was **not** caught, the result from the ***assertTest*** method is returned.

If the ***$exceptionCallback*** class property is set, the specified callback function is called to handle the exception. Otherwise,

- the ***$exceptionCaught*** class property is set to **true**,
- a new ***exceptionDescriptor*** class instance is created to record the ***$exception*** class information; and
- the ***assertLogMessage*** method is called to output a report of the exception if ***$reportFailures*** is set to true.

A **false** value is returned, indicating a test failure.

assertTrue

Perform an assertion that the test is true.

Algorithm

1. Accept ***assertion*** as a string containing the assertion to test;
2. accept ***message*** as a string containing the string to print;
3. pass ***assertion***, ***message*** and a **true** value to the **assertTest** class method and exit with the returned result.

Implementation

```
/**
 * assertTrue
 *
 * perform an assertion that the test is true
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null = none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertTrue($assertion, $message=null)
{
	return $this->assertTest($assertion, $message, true);
}
```

Narrative

The ***assertTrue*** method expects a single, mandatory parameter:

Parameter	Description
$assertion	The test assertion to be executed.

and a single optional parameter:

Parameter	Description
$message	An assert test message to output prior to processing.

The ***assertTest*** method is called with the ***$assertion*** and ***$message*** parameters, and a **true** value for the ***assertTest*** method ***$expected*** parameter.

The result of the ***assertTest*** method is returned.

assertFalse

Perform an assertion that the test is false.

Algorithm

1. Accept ***assertion*** as a string containing the assertion to test;
2. accept ***message*** as a string containing the string to print;
3. pass ***assertion***, ***message*** and a **false** value to the **assertTest** class method and exit with the returned result.

Implementation

```
/**
 * assertFalse
 *
 * perform an assert that the test is false
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null for none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertFalse($assertion, $message=null)
{
	return $this->assertTest($assertion, $message, false);
}
```

The ***assertFalse*** method expects a single, mandatory parameter:

Parameter	Description
$assertion	The test assertion to be executed.

and a single optional parameter:

Parameter	Description
$message	An assert test message to output prior to processing.

The ***assertTest*** method is called with the ***$assertion*** and ***$message*** parameters, and a **false** value for the ***assertTest*** method ***$expected*** parameter.

The result of the ***assertTest*** method is returned.

assertExceptionTrue

Perform an assert that the test is **true**, returns **false** on exception or false assertion.

Algorithm

1. Accept ***assertion*** as a string containing the assertion to test;
2. accept ***message*** as a string containing the message to print;
3. pass ***assertion***, ***message*** and a **true** value to the **assertException** class method;
4. if the result is **true** goto step **6**;
5. if the ***exceptionCaught*** class property is **false**, goto step **7**;
6. exit with a **true** value;
7. exit with a **false** value.

Implementation

```
/**
 * assertExceptionTrue
 *
 * perform an assert that the test is true, otherwise
 *     return false on exception or false assertion
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null for none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertExceptionTrue($assertion, $message=null)
{
      if ($this->assertException($assertion, $message, true) ||
          $this->exceptionCaught)
      {
            return true;
      }

      return false;
}
```

The ***assertExceptionTrue*** method expects a single, mandatory parameter:

Parameter	Description
$assertion	The test assertion to be executed.

and a single optional parameter:

Parameter	Description
$message	An assert test message to output prior to processing.

The ***assertException*** method is called with the ***$assertion*** and ***$message*** parameters, and a **true** value for the ***assertException*** method ***$expected*** parameter.

If the ***assertException*** method returns true, or the ***$exceptionCaught*** class property is true, a **true** value is returned.

Otherwise, a **false** value is returned.

assertExceptionFalse

Perform an assert that the test is **false**, returns **true** on exception or false assertion.

Algorithm

1. Accept ***assertion*** as a string containing the assertion to test;
2. accept ***message*** as a string containing the message to print;
3. pass ***assertion***, ***message*** and a **false** value to the **assertException** class method;
4. if the result is **true** goto step **8**;
5. if the ***exceptionCaught*** class property is **false**, goto step **9**;
6. if the ***reportFailures*** class property is **false**, goto step **8**;
7. pass a string containing concatenated string '**Assertion caught** ' with the ***exception*** class property to the **assertLogMessage** class method;
8. exit with a **true** value;
9. exit with a **false** value.

Implementation

```
/**
 * assertExceptionFalse
 *
 * perform an assert that the test is false,
 *     return false if assert is true or on exception
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null = none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertExceptionFalse($assertion, $message=null)
{
	if ($this->assertException($assertion, $message, false))
	{
		return true;
	}

	if ($this->exceptionCaught)
	{
		if ($this->reportFailures)
		{
			$this->assertLogMessage(sprintf("***  Assertion caught %s  ***",
                                                $this->exception));
		}

		return true;
	}

	return false;
}
```

Narrative

The ***assertExceptionFalse*** method expects a single, mandatory parameter:

Parameter	Description
$assertion	The test assertion to be executed.

and a single optional parameter:

Parameter	Description
$message	An assert test message to output prior to processing.

The ***assertException*** method is called with the ***$assertion*** and ***$message*** parameters, and a **false** value for the ***assertException*** method ***$expected*** parameter.

If the ***assertException*** method returns **true**, or the ***$exceptionCaught*** class property is **true**, ***assertLogMessage*** is called to output a diagnostic message. A **true** value is returned.

Otherwise, a **false** value is returned.

getException

Return the current exception object.

Algorithm

1. Exit with the ***exception*** class property.

Implementation

```
/**
 * getException
 *
 * return the current exception
 * @return \Library\Exception\Descriptor or null if not assigned.
 */
public function getException()
{
       return $this->exception;
}
```

Narrative

Returns the current setting of the ***$exception*** class property.

getExceptionCode

Return the code associated with the last exception.

Algorithm

1. If the ***exception*** class property is not null, goto step **3**;
2. exit, returning **null**;
3. exit, returning the ***code*** property of the ***exception*** class property.

Implementation

```
/**
 * getExceptionCode
 *
 * get the last exception code
 * @return integer $exceptionCode
 */
public function getExceptionCode()
{
	if (! $this->exception)
	{
		return null;
	}

	return $this->exception->code;
}
```

Narrative

If the ***$exception*** class property is not set, a **null** value is returned. Otherwise, the exception code is returned.

getExceptionMessage

Returns the message associated with the last exception.

Algorithm

1. If the ***exception*** class property is **false**, goto step **3**;
2. exit with the ***message*** property of the ***exception*** class property;
3. exit with a **null** value.

Implementation

```
/**
 * getExceptionMessage
 *
 * get the last exception message
 * @return string $exceptionMessage
 */
public function getExceptionMessage()
{
	if (! $this->exception)
	{
		return null;
	}

	return $this->exception->message;
}
```

Narrative

If the ***$exception*** class property is not set, a **null** value is returned. Otherwise, the ***$message*** property of the ***$exception*** class property is returned.

getExceptionClass

Return the class name of the last exception.

Algorithm

1. If the ***exception*** class property is **null**, exit with a **null** value;
2. exit with the ***className*** property of the ***exception*** class property.

Implementation

```
/**
 * getExceptionClass
 *
 * get the exception class name
 * @return string $exceptionClass
 */
public function getExceptionClass()
{
	if (! $this->exception)
	{
		return null;
	}

	return $this->exception->className;
}
```

Narrative

If the ***$exception*** class property is not set, a **null** value is returned. Otherwise, the exception class name is returned.

exceptionCaught

Return the ***$exceptionCaught*** class property.

Algorithm

1. If the ***exception*** class property is **null**, exit with a **null** value;
2. exit with the ***className*** property of the ***exception*** class property.

Implementation

```
/**
 * exceptionCaught
 *
 * get the current exceptionCaught flag setting
 * @return boolean $exceptionCaught
 */
public function exceptionCaught()
{
	return $this->exceptionCaught;
}
```

Narrative

Returns the current value in the ***$exceptionCaught*** class property.

labelBlock

Output a separator block containing a label.

Algorithm

1. Accept ***label*** as a string containing the label to put in the separator block;
2. accept ***blockLength*** as an optional integer containing the width in bytes of the block;
3. accept ***blockChars*** as a string containing the block characters to use for the separator block;
4. if the ***verbose*** class property is **false**, goto step **15**;
5. if the ***labelBlock*** class property is **false**, goto step **15**;
6. if the ***label*** class property is **false**, goto **15**;
7. if ***blockLength*** is less than **10**, set blockLength to **10**;
8. if ***blockChars*** is null, set ***blockChars*** to a string containing '*';
9. create a string of ***blockLength*** repetitions of ***blockChars*** and assign the string to ***block***;
10. pass ***block*** to **assertLogMessage** class method;
11. pass ***blockChars*** to **assertLogMessage** class method;
12. pass ***blockChars*** concatenated to ***label*** to **assertLogMessage** class method;
13. pass ***blockChars*** to **assertLogMessage** class method;
14. pass ***block*** to **assertLogMessage** class method;
15. exit.

Implementation

```
/**
 * labelBlock
 *
 * Output a separator block containing a label
 * @param string $label = label to put in the separator block
 * @param integer $blockLength = (optional) width of the block in length
 *                  ($blockChars) bytes (must be greater than 9) (default = 10)
 * @param string $blockChars = (optional) character(s) to use for the separator block
 */
public function labelBlock($label, $blockLength=10, $blockChars='*')
{
	if ((! $this->verbose) || (! $this->labelBlock) || (! $label))
	{
		return;
	}

	if ($blockLength < 10)
	{
		$blockLength = 10;
	}

	if (! $blockChars)
	{
		$blockChars = '*';
	}

	$block = str_repeat($blockChars, $blockLength);
```

```
        $this->assertLogMessage($block);
        $this->assertLogMessage($blockChars);
        $this->assertLogMessage(sprintf("%s\t\t%s", $blockChars, $label));
        $this->assertLogMessage($blockChars);
        $this->assertLogMessage($block);
    }
```

Narrative

The **labelBlock** method expects a single, mandatory parameter:

Parameter	Description
$label	The label to put in the separator block.

and a two optional parameters:

Parameter	Description
$blockLength	Width of the block in length (***$blockChars***) bytes.
$blockChars	character(s) used for the separator block border.

If the ***$verbose*** or ***$labelBlock*** class properties, or the ***$label*** parameter, evaluate to false, no action is taken.

The ***$blockLength*** is set to 10 if it evaluates to less than 10, and the ***$blockChars*** parameter is set to '*', if it is empty, and the ***$block*** string is set to ***$blockLength*** repetitions of the ***$blockChars*** sequence.

The label block is created with the specified ***$label*** parameter.

labelBlockFlag

Set/get the ***$labelBlock*** flag.

Algorithm

1. Accept ***labelBlock*** as a boolean flag: true to set the ***labelBlock*** class property, **false** to reset and **null** to query;
2. if ***labelBlock*** is **null**, goto step **4**;
3. set the ***labelBlock*** class property to ***labelBlock***;
4. exit, returning the ***labelBlock*** class property.

Implementation

```
/**
 * labelBlockFlag
 *
 * set/get label-block flag
 * @param boolean $labelBlock = (optional) true = set, false = reset, null to query
 * @return boolean $labelBlock
 */
public function labelBlockFlag($labelBlock=null)
{
	if ($labelBlock !== null)
	{
		$this->labelBlock = $labelBlock;
	}

	return $this->labelBlock;
}
```

Narrative

The ***labelBlockFlag*** method expects a single, optional parameter:

Parameter	Description
$labelBlock	True to set, false to reset, null to query.

If the ***$labelBlock*** parameter is not null, the ***$labelBlock*** class property is set to the value in the ***$labelBlock*** parameter.

The current value of the ***$labelBlock*** class property is returned.

assertCallback

Assert test callback function.

Algorithm

1. Accept ***script*** as a string containing the name of the script;
2. accept ***line*** as an integer containing the script line number;
3. accept ***message*** as an (optional) assert message;
4. set ***self*** to the static class property ***me***;
5. if the ***reportFailures*** class property is **false**, goto step **7**;
6. create a string by concatenating ***script***, ***line*** and ***message*** and passing the result to the **assertLogMessage** class method;
7. create an array containing '**script**' for a key and ***script*** as the associated field;
8. add an array entry with '**line**' for a key and ***line*** as the associated field;
9. add an array entry with '**message**' for a key and ***message*** as the associated field;
10. assign the array to the ***assertEvent*** class property;
11. exit with a **false** value.

Implementation

```
/**
 * assertCallback
 *
 * Assert test callback function to handle failures, if ASSERT_WARNING != 0
 * @param string $script = script name
 * @param integer $line = line number
 * @param string $message = assert message (or null)
 * @return boolean false
 */
public static function assertCallback($script, $line, $message=null)
{
	$self = self::$me;

	if ($self->reportFailures)
	{
		$self->assertLogMessage(sprintf("Assert failure in\n" .
			"\tScript:\t%s\n\tLine:\t%s\n\tCondition:\t%s\n",
			$script,
			$line,
			$message));
	}

	$self->assertEvent = array('script'	=> $script,
				'line'	=> $line,
				'message'	=> $message);

	return false;
}
```

Narrative

The ***assertCallback*** static method expects two, mandatory parameters:

Parameter	Description
$script	Script name.
$line	Script line number.

and a single optional parameter:

Parameter	Description
$message	The assert message.

The ***$self*** local variable is set to the ***$me*** static class property.

If the ***$reportFailures*** class property is set, a diagnostic message is output by calling the ***assertLogMessage*** method.

The ***$assertEvent*** class array is set to the values of the parameters passed to the method, and a false value is returned.

properties

Get/set the properties object.

Algorithm

1. Accept ***properties*** as an (optional) **Library\Properties** class object to store, **null** to query;
2. if ***properties*** is **null**, goto step **4**;
3. assign ***properties*** to the ***properties*** class property;
4. exit with the ***properites*** class property.

Implementation

```
/**
 * properties
 *
 * Set/get the properties object
 * @param object $properties = (optional) properties object to store,
 *                                 null to query only
 * @return object $properties
 */
public function properties($properties=null)
{
	if ($properties !== null)
	{
		$this->properties = $properties;
	}

	return $this->properties;
}
```

Narrative

The ***properties*** method expects a single, optional parameter:

Parameter	Description
$properties	True to set, false to reset, null to query.

If the ***$properties*** parameter is not null, the ***$properties*** class property is set to the value in the ***$properties*** parameter.

The current value of the ***$properties*** class property is returned.

verbose

Get/set the ***$verbose*** setting.

Algorithm

1. Accept ***verbose*** as an (optional) integer containing the verbosity setting;
2. if ***verbose*** is null, goto step **4**;
3. assign ***verbose*** to the ***verbose*** class property;
4. exit with the ***verbose*** class property.

Implementation

```
/**
 * verbose
 *
 * set/get verbosity setting
 * @param integer $verbose = (optional) verbose setting
 *                           (0 = silent, non-zero = output ok), null to query only
 * @return integer $verbose
 */
public function verbose($verbose=null)
{
	if ($verbose !== null)
	{
		$this->verbose = $verbose;
	}

	return $this->verbose;
}
```

Narrative

The ***verbose*** method expects a single, optional parameter:

Parameter	Description
$verbose	0 = silent, 1 = minimal messages, 2 = detailed.

If the ***$verbose*** parameter is not null, the ***$verbose*** class property is set to the value in the ***$verbose*** parameter.

The current value of the ***$verbose*** class property is returned.

errorsOnly

Display only errors flag, used in assertTest to limit output.

Algorithm

1. Accept ***errorsOnly*** as an (optional) integer containing the errors only setting;
2. if ***errorsOnly*** is null, goto step 4;
3. assign ***errorsOnly*** to the ***errorsOnly*** class property;
4. exit, returning the ***errorOnly*** class property.

Implementation

```
/**
 * errorsOnly
 *
 * Errors only flag: 0 = display all messages, <> 0 = don't display errors
 * @params integer $errorsOnly = (optional) errors only flag, null to query only
 * @return integer $errorsOnly
 */
public function errorsOnly($errorsOnly=null)
{
	if ($errorsOnly !== null)
	{
		$this->errorsOnly = $errorsOnly;
	}

	return $this->errorsOnly;
}
```

Narrative

The ***errorsOnly*** flag is only used in the ***assertTest*** method to limit the amount of output generated when ***$verbose*** is non-zero.

The ***errorsOnly*** method expects a single, optional parameter:

Parameter	Description
$errorsOnly	0 = output all messages, non-zero = limit messages.

If the ***$errorsOnly*** parameter is not null, the ***$errorsOnly*** class property is set to the value in the ***$errorsOnly*** parameter.

The current value of the ***$errorsOnly*** class property is returned.

testNumber

Get/set the ***$testNumber*** class property.

Algorithm

1. Accept ***testNumber*** as an (optional) integer containing the test number;
2. if ***testNumber*** is **null**, goto step **4**;
3. assign ***testNumber*** to the ***testNumber*** class property;
4. exit, returning the ***testNumber*** class property.

Implementation

```
/**
 * testNumber
 *
 * get/set test number
 * @param integer $testNumber = (optional) test number, null to query only
 * @return integer $testNumber
 */
public function testNumber($testNumber=null)
{
	if ($testNumber !== null)
	{
		$this->testNumber = $testNumber;
	}

	return $this->testNumber;
}
```

Narrative

The ***testNumber*** method expects a single, optional parameter:

Parameter	Description
$testNumber	The current sub-test number, null to query.

If the ***$testNumber*** parameter is not null, the ***$testNumber*** class property is set to the value in the ***$testNumber*** parameter.

The current value of the ***$testNumber*** class property is returned.

testName

Get/set the test name.

Algorithm

1. Accept ***testName*** as an (optional) string containing the name of the test;
2. if ***testName*** is **null**, goto step **4**;
3. assign ***testName*** to the ***testName*** class property;
4. exit, returning the ***testName*** class property.

Implementation

```
/**
 * testName
 *
 * get/set test name
 * @param string $testName = (optional) test name, null to query only
 * @return string $testName
 */
public function testName($testName=null)
{
	if ($testName !== null)
	{
		$this->testName = $testName;
	}

	return $this->testName;
}
```

Narrative

The ***testName*** method expects a single, optional parameter:

Parameter	Description
$testName	The test name, null to query.

If the ***$testName*** parameter is not null, the ***$testName*** class property is set to the value in the ***$testName*** parameter.

The current value of the ***$testName*** class property is returned.

assertEventArray

Get the event array.

Algorithm

1. exit, returning the ***assertEvent*** class property.

Implementation

```
/**
 * assertEventArray
 *
 * get the array containing information about the last assert failure
 * @return array $assertEvemt
 */
public function assertEventArray()
{
       return $this->assertEvent;
}
```

Narrative

The current setting of the ***$assertEvent*** class property is returned.

assertLogMessage

Output a formatted message to the current output device(s).

Algorithm

1. Accept ***message*** as a string containing the message to output;
2. accept ***level*** as an (optional) string containing the debug level;
3. if ***verbose*** is **0**, goto step **10**;
4. create an array containing '**level**' as the key and ***level*** as the associated field;
5. add an array entry with '**program**' as the key and the ***testName*** class property as the associated field;
6. add an array entry with '**method**' as the key and the current method name as the associated field;
7. pass the current class instance to the **getEldestParent** class method, and add as the field for an array entry with '**class**' as the key;
8. add an array entry with '**skiplevels**' as the key and **7** as the associated field;
9. pass a string containing the ***testNumber*** class property and ***message*** concatenated, and the array to the **message** class method of the **Library\Log** class;
10. exit.

Implementation

```
/**
 * assertLogMessage
 *
 * Output a message + newline to the current output device
 * @param string $message = message to output
 * @return null
 */
public function assertLogMessage($message, $level='debug')
{
      if ($this->verbose)
      {
            Log::message(sprintf("Subtest #%u - %s", $this->testNumber, $message),
                                 array("level"      => $level,
                                       "program"    => $this->testName,
                                       "method"     => substr(__METHOD__,
                                                         strpos(__METHOD__, "::") + 2),
                                       "class"      => $this->getEldestParent($this),
                                       "skiplevels" => 7));
      }
}
```

Narrative

The ***assertLogMessage*** method expects a single, mandatory parameter:

Parameter	Description
$message	The message to output.

and a single optional parameter:

Parameter	Description
$level	The log-level of the message.

Refer to the ***Log\Format::log*** method documentation for details on the ***$message*** and ***$level*** parameters, and log message formatting requirements.

assertFailures

Get/set the reportFailures flag.

Algorithm

1. Accept ***report*** as an (optional) boolean containing the ***reportFailures*** class property setting, or **null** to query;
2. if ***report*** is null, goto step **4**;
3. assign ***report*** to the ***reportFailures*** class property;
4. exit, returning the ***reportFailures*** class property.

Implementation

```
/**
 * assertFailures
 *
 * True to report assert failures, false to not, regardless of $verbose
 * @param boolean $report = (optional) report setting, null to query
 * @return boolean $report
 */
public function assertFailures($report=null)
{
	if ($report !== null)
	{
		$this->reportFailures = $report;
	}

	return $this->reportFailures;
}
```

Narrative

The ***assertFailures*** method expects a single, optional parameter:

Parameter	Description
$assertFailures	True to override the ***$verbose*** class property on an assert failure.

If the ***$assertFailures*** parameter is not null, the ***$assertFailures*** class property is set to the value in the ***$assertFailures*** parameter.

The current value of the ***$assertFailures*** class property is returned.

The ***$reportFailures*** class property overrides the setting of the ***$verbose*** class property when an assert failure happens.

assertExceptionDescriptor

Get a copy of the exception descriptor.

Algorithm

1. exit, returning the ***exception*** class property.

Implementation

```
/**
 * assertExceptionDescriptor
 *
 * get a copy of the exception object
 * @return \Library\Exception\Descriptor
 */
public function assertExceptionDescriptor()
{
	return $this->exception;
}
```

Narrative

The current setting of the ***$exception*** class property is returned.

getEldestParent

Get the top parent in the hierarchy.

Algorithm

1. Accept ***object*** as the name of the class or an object instance to get the eldest parent for;
2. if ***object*** is not an class instance, goto step 4;
3. assign the name of the class instance to ***class***;
4. if ***object*** not a string, goto step **7**;
5. assign ***object*** to ***class***;
6. goto step **9**;
7. create a new **Library\Testing\Exception** class instance and pass the '**StringOrObjectExpected**' constant to the class;
8. throw the exception and exit;
9. if the parent of ***class*** is null, goto step **12**;
10. assign the parent of ***class*** to ***parent***;
11. goto step **9**;
12. exit, returning ***class***.

Implementation

```
/**
 * getEldestParent
 *
 * Get the top parent in the hierarchy
 * @param mixed $object = class name or object to get eldest parent for
 * @return string $class = eldest parent, false if no parent
 * @throws Library\Testing\Exception
 */
public function getEldestParent($object)
{
	if (is_object($object))
	{
		$class = get_class($object);
	}
	elseif (is_string($object))
	{
		$class = $object;
	}
	else
	{
		throw new Exception('StringOrObjectExpected');
	}

	while($parent = get_parent_class($class))
	{
		$class = $parent;
	}

	return $class;
}
```

Narrative

The ***getEldestParent*** method expects a single, mandatory parameter:

Parameter	Description
$object	The class name or class object to get the eldest parent for.

If the ***$object*** parameter is an object, the PHP **get_class function** is called to convert the ***$object*** parameter to a class name string. Otherwise, if the object is a string, the ***$object*** parameter is assigned to the ***$class*** class name.

If the ***$object*** parameter is neither an object nor a string, a ***Testing\Exception*** is thrown.

The PHP **get_parent_class** is repeatedly called on the current ***$class*** method variable to retrieve the name of the parent class to the ***$parent*** method variable, and store it in the ***$class*** method variable, until a **false** value is returned to ***$parent***.

The final, valid ***$class*** method variable is returned.

assertExceptionCallback

Get/set the exception callback function.

Algorithm

1. Accept ***callback*** as an (optional) string containing the name of the callback method in the current class, or **null** to query;
2. if ***callback*** is **null**, goto step **4**;
3. if the callback method exists, assign ***callback*** to the ***exceptionCallback*** class property;
4. exit, returning the ***exceptionCallback*** class property.

Implementation

```
/**
 * assertExceptionCallback
 *
 * Set/get exception callback function
 * @param string $callback = name of the callback function
 */
public function assertExceptionCallback($callback=null)
{
	if ($callback !== null)
	{
		if (method_exists($this, $callback))
		{
			$this->exceptionCallback = $callback;
		}
	}

	return $this->exceptionCallback;
}
```

Narrative

The ***assertExceptionCallback*** expects a single, optional parameter:

Parameter	Description
$callback	The name of the callback function.

If the ***$callback*** parameter is not null, and the ***$callback*** is a valid class method in the current test class, the ***$exceptionCallback*** class property is set to the ***$callback*** parameter.

The current ***$exceptionCallback*** class property is returned.

assertStartLogger

Create a test-relative log file and start the logger.

Algorithm

1. Accept ***logName*** as an (optional) string containing the internal name of the logging device;
2. if the **assertLogger** class property is **null**, goto step **5**;
3. if the **assertLogger** class property is not equal to ***logName***, goto step **5**;
4. pass ***logName*** to the **assertStopLogger** class method;
5. pass the result of calling the **getLogDefaults** method of the **setup** class in the ***properties*** class property to the **constructor** method for the **Library\Properties** class and assign the result to ***loggerProperties***;
6. assign ***logName*** to the ***Log_Name*** property of the ***loggerProperties*** class;
7. assign the '**fileio**' string constant to the ***Log_Adapter*** property of the ***loggerProperties*** class;
8. assign the '**debug**' string constant to the ***Log_Level*** property of the ***loggerProperties*** class;
9. replace all "\\" and "_" characters in the ***testName*** class property string with "**-**" characters and assign the result to ***logFileName***;
10. if the first character in ***logFileName*** is not equal to "**-**", goto step **12**;
11. delete the first character in ***logFileName***;
12. concatenate the directory name of the ***Log_FileDestination*** property of the ***properties*** class property with the PHP **DIRECTORY_SEPARATOR** and the ***logFileName*** and store the result in the ***Log_FileDestination*** property of the ***loggerProperties*** class;
13. assign the '**fileobject**' string constant to the ***Log_FileAdapter*** property of the ***loggerProperties*** class;
14. assign the '**w**' string constant to the ***Log_FileMode*** property of the ***loggerProperties*** class;
15. create a PHP **try** block;
16. pass ***loggerProperties*** to the ***startLog*** property of the **Library\Log** class;
17. end the PHP **try** block;
18. create a PHP **catch** block to catch a **Library\Log\Exception**;
19. assign the ***exception*** to ***exception***;
20. concatenate the string '**Unable to open disk file log:** ' with the result of calling the **getMessage** method of the ***exception*** class and pass the result to the **message** method of the **Library\Log** class;
21. pass the string '**Logging to disk file disabled**' to the **message** method of the **Library\Log** class;
22. exit, returning a **false** value;
23. end the PHP **catch** block;
24. assign ***logName*** to the ***assertLogger*** class property;
25. concatenate '**Test Program:** ' with the ***testName*** class property and pass the result to the **assertLogMessage** class method;
26. if the **parameterCount** method of the **Library\CliParameters** class returns a non-zero result, goto step **29**;
27. call the **parameters** method of the **Library\CliParameters** class, pass the result to the **format** method of the **Library\PrintU\FormatArray** class and

pass the result to the **assertLogMessage** class method;
28. goto step **30**;
29. pass the string '**No CliParameters**' to the **assertLogMessage** class method;
30. assign the date and time to the ***Test_Start*** property of the ***properties*** class property;
31. assign **null** to the ***Test_End*** property of the ***properties*** class property;
32. pass the ***Test_Start*** property of the ***properties*** class property to the **assertLogMessage** class method;
33. exit, returning a **true** value.

Implementation

```
/**
 * assertStartLogger
 *
 * Create a test-relative log file and start the logger
 * @param string $logName = (optional) internal logger name, default = 'testlogger'
 * @return boolean $result = true if successful, false if not
 */
public function assertStartLogger($logName='testlogger')
{
	if ($this->assertLogger && ($this->assertLogger == $logName))
	{
		$this->assertStopLogger($logName);
	}

	$loggerProperties =
		new \Library\Properties($this->properties->setup->getLogDefaults());

	$loggerProperties->Log_Name      = $logName;
	$loggerProperties->Log_Adapter   = 'fileio';
	$loggerProperties->Log_Level     = 'debug';

	$logFileName = str_replace(array('\\', '_'), '-', $this->testName);
	if (substr($logFileName, 0, 1) == '-')
	{
		$logFileName = substr($logFileName, 1);
	}

	$loggerProperties->Log_FileDestination =
		dirname($this->properties['Log_FileDestination']) .
		DIRECTORY_SEPARATOR . $logFileName;
	$loggerProperties->Log_FileAdapter     = 'fileobject';
	$loggerProperties->Log_FileMode        = 'w';

	try
	{
		Log::startLog($loggerProperties);
	}
	catch(\Library\Log\Exception $exception)
	{
		Log::message(sprintf('Unable to open disk file log: %s',
				$exception->getMessage()), 'error');
		Log::message('Logging to disk file has been disabled', 'error');
		return false;
	}

	$this->assertLogger = $logName;
```

```
        $this->assertLogMessage(sprintf('Test Program: %s', $this->testName));

        if (CliParameters::parameterCount() == 0)
        {
              $this->assertLogMessage('    NO CLI Parameters');
        }
        else
        {
              $this>assertLogMessage
                    (\Library\PrintU\FormatArray::format(CliParameters::parameters(),
                     'CLI Parameters', false, false, false));
        }

        $this->properties->Test_Start = date('Y-m-d H:i:s') .
                                        substr((string)microtime(), 1, 6);
        $this->properties->Test_End   = null;
        $this->assertLogMessage('Start-of-test @ ' . $this->properties->Test_Start);
        return true;
  }
```

The ***assertStartLogger*** method expects a single, optional parameter:

Parameter	Description
$logName	The internal logger name.

If the ***$assertLogger*** class property is set, and it is the same name as the ***$logName*** parameter, the ***assertStopLogger*** method is called to stop the named logger. Otherwise, the ***$logName*** is set to the default value of '**testlogger**'.

The ***$loggerProperties*** class property is set to a new ***Properties*** class instance and initialized by calling the ***Testing\Setup::getLogDefaults*** method.

The ***Log_Name***, ***Log_Adapter*** and ***Log_Level*** variables are initialized in the ***$loggerProperties***, as shown.

The ***Log_FileDestination***, ***Log_FileAdapter*** and ***Log_FileMode*** variable are also initialized in the ***$loggerProperties***, also as shown.

A new logger is started by calling the ***Log::startLog*** method and passing it the ***$loggerProperties*** object. If the ***Log::startLog*** method throws an exception, the logger is disabled and a **false** value is returned.

The ***$assertLogger*** class property is set to the log file name and the name of the test class and any test class-relative CLI parameters are output to the logger by calling the ***assertLogMessage*** method.

Finally, the ***Test_Start*** time and date is set in the ***$properties*** class property, and the '***Start-of-test***' message is output to the logger by calling the ***assertLogMessage*** method.

A **true** result is returned.

assertStopLogger

Stop the current logger, if it is running.

Algorithm

1. Accept ***logName*** as an (optional) string containing the internal log name to stop, null to stop the current logger;
2. increment the ***testNumber*** class property;
3. set the ***Test_End*** property of the ***properties*** class property to the current date and time;
4. concatenate the '**End-of-test** ' string and the ***Test_End*** property of the ***properties*** class property and pass the result to the **assertLogMessage** class method;
5. if ***logName*** is not **null**, goto step **7**;
6. assign the ***assertLogger*** class property to ***logName***;
7. pass ***logName*** to the **stopLog** method of the **Library\Log** class;
8. if ***logName*** is not equal to the ***assertLogger*** class property, goto step **10**;
9. assign **null** to the ***assertLogger*** class property;
10. exit.

Implementation

```
/**
 * assertStopLogger
 *
 * Stop the current logger, if it is running
 * @param string $logName = (optional) internal log name to stop,
 *                                     null to stop current logger
 */
public function assertStopLogger($logName=null)
{
	$this->testNumber++;

	$this->properties->Test_End = date('Y-m-d H:i:s') .
	                              substr((string)microtime(), 1, 6);

	$this->assertLogMessage('End-of-test @ ' . $this->properties->Test_End);

	if ($logName == null)
	{
		$logName = $this->assertLogger;
	}

	Log::stopLog($logName);

	if ($logName == $this->assertLogger)
	{
		$this->assertLogger = null;
	}
}
```

Narrative

The ***assertStopLogger*** method expects a single, mandatory parameter:

Parameter	Description
$logName	The name of the internal log to stop.

If the ***$logName*** parameter is null, the current ***$assertLogger*** class property is used.

The ***$testNumber*** class property is incremented and the ***Test_End*** variable in the ***$properties*** class property is set to the current time.

The ***Log::stopLog*** method is passed the ***$logName*** to stop the log.

If the ***$logName*** parameter is the same name as the ***$assertLogger*** class property, the ***$assertLogger*** variable is set to null.

Source Listing

```
<?php
namespace Library\Testing;
use Library\Log;
use Library\Exception\Descriptor;
use Library\CliParameters;

/*
 *     Testing\Base 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
*/
/**
 * Testing\Base
 *
 * Testing\Base class for assertion test classes
 *           provides the assert functions and support methods
 * @author Jay Wheeler
 * @version 1.0
 * @copyright © 2012, 2013 EarthWalk Software.
 * @license Licensed under the Academic Free License version 3.0.
 * @package Testing
 * @subpackage Base
 */
class Base
{
       /**
        * properties
        *
        * Properties class instance
        * @var object $properties
        */
       protected                $properties;

       /**
        * exceptionCaught
        *
        * true = exception caught in last assert, false = exception not caught
        * @var boolean $exceptionCaught
        */
       protected                $exceptionCaught;

       /**
        * exception
        *
        * @var Library\Exception\Descriptor
        */
       protected                $exception;

       /**
        * exceptionCallback
        *
        * contains the name of a user-defined callback processing method
        * @var string $exceptionCallback
        */
       protected                $exceptionCallback;
```

```
/**
 * verbose
 *
 * output: short messages when 1, detailed message when 2, no messages when 0
 * @var integer $verbose
 */
protected                 $verbose;

/**
 * reportFailures
 *
 * True to report assert failures, even if not Verbose.
 * False to inhibit assert faiures, even if Verbose.
 * @var boolean $reportFailures
 */
protected                 $reportFailures;

/**
 * assertLogger
 *
 * name of the assert logger internal log name
 * @var string $assertLogger
 */
protected                 $assertLogger;

/**
 * loggerProperties
 *
 * A Library\Properties object for by-test logging properties
 * @var object $loggerProperties
 */
protected                 $loggerProperties;

/**
 * errorsOnly
 *
 * 0 = all messages, 1 = errors only
 * @var integer $errorsOnly
 */
protected                 $errorsOnly;

/**
 * testNumber
 *
 * The current subTest number
 * @var integer $testNumber
 */
protected                 $testNumber;

/**
 * testName
 *
 * The name of the test
 * @var string $testName
 */
protected                 $testName;
```

```
/**
 * assert
 *
 * An array containing information about the last false assertion processed
 * @var array $assert
 */
protected                 $assert;

/**
 * labelBlock
 *
 * True if ok to output a block label, false to not
 * @var boolean $labelBlock
 */
protected                 $labelBlock;

/**
 * cliParameters
 *
 * Library\CliParameters instance containing test program cli parameters
 * @var object $cliParameters
 */
protected                 $cliParameters = null;

/**
 * eolSequence
 *
 * The current end-of-line sequence
 * @var string $eolSequence
 */
protected                 $eolSequence = '';

/**
 * me
 *
 * This class instance
 * @var object $me
 */
protected static    $me;

/**
 * __construct
 *
 * Class constructor
 */
public function __construct()
{
       $this->properties = null;
       $this->loggerProperties = null;

       $this->exception = null;
       $this->exceptionCaught = false;
       $this->exceptionCallback = null;

       $this->assertEvent = array();

       $this->verbose = 0;
       $this->errorsOnly = 1;
       $this->testNumber = 0;
       $this->testName = '<unknown>';
       $this->eolSequence = null;
```

```
        $this->reportFailures = true;
        $this->labelBlock = true;

        \Library\Autoload::loadClass('\Library\CliParameters');

        self::$me = $this;
}

/**
 * __destruct
 *
 * Destroy the current class instance
 */
public function __destruct()
{
}

/**
 * assertSetup
 *
 * Set assert options
 * @param integer $active   = (optional) active flag: 0 ==> inactive, 1 ==> active
 *                               (Default: 1)
 * @param integer $warning  = (optional) warning flag: 0 ==> silent, 1 ==> issue
 *                                warning on assert failure. (Default: 0)
 * @param integer $bail     = (optional) bail-out flag: 0 ==> keep processing,
 *                                            1 ==> abort processing. (Default: 0)
 * @param string  $callback = (optional) callback function name.
 *                          (Default = \Library\Testing\Base::assertCallback()).
 */
protected function assertSetup($active=1, $warning=0, $bail=0,
                               $callback=array('\Library\Testing\Base',
                                               'assertCallback'))
{
        assert_options(ASSERT_ACTIVE,    $active);
        assert_options(ASSERT_WARNING,   $warning);
        assert_options(ASSERT_BAIL,             $bail);
        assert_options(ASSERT_CALLBACK,  $callback);
}
```

```
/**
 * assertTest
 *
 * Assert the assertion and return the result conditioned by $expected
 * @param string $assertion = assertion to test
 * @param string $message = (optional) string to print if $verbose is true,
 *                                     null for no message
 * @param boolean $expected = (optional) expected result
 *                                    (true = a true result WAS expected;
 *                                     false = a false result WAS expected)
 * @return boolean true = result was as expected, false = it was not as expected
 */
protected function assertTest($assertion, $message=null, $expected=true)
{
	++$this->testNumber;

	if ($this->verbose && ($message !== null))
	{
		$this->assertLogMessage($message);
	}

	if (! assert($assertion))
	{
		if ($expected == false)
		{
			if ($this->verbose && (! $this->errorsOnly))
			{
				$this->assertLogMessage("Assertion is false");
			}

			return true;
		}

		$this->assertLogMessage
			("***  Assertion is FALSE but expected TRUE ***");
		return false;
	}

	if ($expected == true)
	{
		if ($this->verbose && (! $this->errorsOnly))
		{
			$this->assertLogMessage("Assertion is true");
		}

		return true;
	}

	$this->assertLogMessage("***  Assertion is TRUE but expected FALSE  ***");
	return false;
}
```

```
/**
 * assertException
 *
 * Assert the assertion and return the result conditioned by $expected
 *                                                    and $exceptionCaught
 * @param string $assertion = assertion to test
 * @param string $message = (optional) string to print if $verbose is true,
                                       null for no message
 * @param boolean $expected = (optional) expected result
 *                                       (true = a true result WAS expected;
 *                                        false = a false result WAS expected)
 * @return boolean true = result was as expected,
 *                        false = it was not as expected, or an exception occurred
 */
public function assertException($assertion, $message=null, $expected=true)
{
      try
      {
            $this->exceptionCaught = false;
            return $this->assertTest($assertion, $message, $expected);
      }
      catch(\Exception $exception)
      {
            if ($this->exceptionCallback)
            {
                  $this->{$this->exceptionCallback}($exception);
            }
            else
            {
                  $this->exceptionCaught = true;
                  $this->exception = new \Library\Exception\Descriptor($exception,

                   array('testname'   => $this->testName,
                         'testnumber' => $this->testNumber));
                  if ($this->reportFailures)
                  {
                        $this->assertLogMessage(sprintf("%s Test #%u - %s",
                                    $this->exception->testname,
                                    $this->exception->testnumber,
                                    $this->exception));
                  }
            }
      }

      return false;
}
```

```
/**
 * assertTrue
 *
 * perform an assertion that the test is true
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null = none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertTrue($assertion, $message=null)
{
      return $this->assertTest($assertion, $message, true);
}

/**
 * assertFalse
 *
 * perform an assert that the test is false
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null for none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertFalse($assertion, $message=null)
{
      return $this->assertTest($assertion, $message, false);
}

/**
 * assertExceptionTrue
 *
 * perform an assert that the test is true, otherwise return false on exception
 *                                                          or false assertion
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null for none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertExceptionTrue($assertion, $message=null)
{
      if ($this->assertException($assertion, $message, true) ||
          $this->exceptionCaught)
      {
            return true;
      }

      return false;
}
```

```
/**
 * assertExceptionFalse
 *
 * perform an assert that the test is false, return false if assert is true
 *                                                    or on exception
 * @param string $assertion = assertion to test
 * @param string $message = assert test message, null = none
 * @return boolean true = successful, false = unsuccessful
 */
public function assertExceptionFalse($assertion, $message=null)
{
	if ($this->assertException($assertion, $message, false))
	{
		return true;
	}

	if ($this->exceptionCaught)
	{
		if ($this->reportFailures)
		{
			$this->assertLogMessage
			 (sprintf("***  Assertion caught EXCEPTION: (%u) %s  ***",
					$this->exception->code,
					$this->exception->message));
			$this->assertLogMessage((string)$this->exception);
		}

		return true;
	}

	return false;
}

/**
 * getException
 *
 * return the current exception
 * @return \Library\Exception\Descriptor or null if not assigned.
 */
public function getException()
{
	return $this->exception;
}

/**
 * getExceptionCode
 *
 * get the last exception code
 * @return integer $exceptionCode
 */
public function getExceptionCode()
{
	if (! $this->exception)
	{
		return null;
	}

	return $this->exception->code;
}
```

```
/**
 * getExceptionMessage
 *
 * get the last exception message
 * @return string $exceptionMessage
 */
public function getExceptionMessage()
{
	if (! $this->exception)
	{
		return null;
	}

	return $this->exception->message;
}

/**
 * getExceptionClass
 *
 * get the exception class name
 * @return string $exceptionClass
 */
public function getExceptionClass()
{
	if (! $this->exception)
	{
		return null;
	}

	return $this->exception->className;
}

/**
 * exceptionCaught
 *
 * get the current exceptionCaught flag setting
 * @return boolean $exceptionCaught
 */
public function exceptionCaught()
{
	return $this->exceptionCaught;
}
```

```
/**
 * labelBlock
 *
 * Output a separator block containing a label
 * @param string $label = label to put in the separator block
 * @param integer $blockLength = (optional) width of the block in
 *                               length($blockChars) bytes
 *                                        (must be greater than 9) (default = 10)
 * @param string $blockChars = (optional) character(s) to use for the
 *                                           separator block (default = "*")
 */
public function labelBlock($label, $blockLength=10, $blockChars='*')
{
	if ((! $this->verbose) || (! $this->labelBlock) || (! $label))
	{
		return;
	}

	if ($blockLength < 10)
	{
		$blockLength = 10;
	}

	if (! $blockChars)
	{
		$blockChars = '*';
	}

	$block = str_repeat($blockChars, $blockLength);

	$this->assertLogMessage($block);
	$this->assertLogMessage($blockChars);
	$this->assertLogMessage(sprintf("%s\t\t%s", $blockChars, $label));
	$this->assertLogMessage($blockChars);
	$this->assertLogMessage($block);
}

/**
 * labelBlockFlag
 *
 * set/get label-block flag
 * @param boolean $labelBlock = (optional) true = set, false = reset, null to query
 * @return boolean $labelBlock
 */
public function labelBlockFlag($labelBlock=null)
{
	if ($labelBlock !== null)
	{
		$this->labelBlock = $labelBlock;
	}

	return $this->labelBlock;
}
```

```
/**
 * assertCallback
 *
 * Assert test callback function to handle failures, if ASSERT_WARNING != 0
 * @param string $script = script name
 * @param integer $line = line number
 * @param string $message = assert message (or null)
 * @return boolean false
 */
public static function assertCallback($script, $line, $message=null)
{
      $self = self::$me;

      if ($self->reportFailures)
      {
            $self->assertLogMessage
                  (sprintf("Assert failure in\n\tScript:\t%s\n\tLine:\t" .
                              "%s\n\tCondition:\t%s\n",
                        $script,
                        $line,
                        $message));
      }

      $self->assertEvent = array('script'  => $script,
                                 'line'    => $line,
                                 'message' => $message);

      return false;
}

/**
 * properties
 *
 * Set/get the properties object
 * @param object $properties = (optional) properties object to store,
 *                                        null to query only
 * @return object $properties
 */
public function properties($properties=null)
{
      if ($properties !== null)
      {
            $this->properties = $properties;
      }

      return $this->properties;
}
```

```
/**
 * verbose
 *
 * set/get verbosity setting
 * @param integer $verbose = (optional) verbose setting (0 = silent,
 *                                      non-zero = output ok), null to query only
 * @return integer $verbose
 */
public function verbose($verbose=null)
{
	if ($verbose !== null)
	{
		$this->verbose = $verbose;
	}

	return $this->verbose;
}

/**
 * errorsOnly
 *
 * Errors only flag: 0 = display all messages, <> 0 = don't display errors
 * @params integer $errorsOnly = (optional) errors only flag, null to query only
 * @return integer $errorsOnly
 */
public function errorsOnly($errorsOnly=null)
{
	if ($errorsOnly !== null)
	{
		$this->errorsOnly = $errorsOnly;
	}

	return $this->errorsOnly;
}

/**
 * testNumber
 *
 * get/set test number
 * @param integer $testNumber = (optinal) test number, null to query only
 * @return integer $testNumber
 */
public function testNumber($testNumber=null)
{
	if ($testNumber !== null)
	{
		$this->testNumber = $testNumber;
	}

	return $this->testNumber;
}
```

```
    /**
     * testName
     *
     * get/set test name
     * @param string $testName = (optinal) test name, null to query only
     * @return string $testName
     */
    public function testName($testName=null)
    {
           if ($testName !== null)
           {
                  $this->testName = $testName;
           }

           return $this->testName;
    }

   /**
    * assertEventArray
    *
    * get the array containing information about the last assert failure
    * @return array $assertEvemt
    */
   public function assertEventArray()
   {
          return $this->assertEvent;
   }

   /**
    * assertFailures
    *
    * True to report assert failures, false to not, regardless of $verbose
    * @param boolean $report = (optional) report setting, null to query
    * @return boolean $report
    */
   public function assertFailures($report=null)
   {
          if ($report !== null)
          {
                 $this->reportFailures = $report;
          }

          return $this->reportFailures;
   }

  /**
   * assertExceptionDescriptor
   *
   * get a copy of the exception object
   * @return \Library\Exception\Descriptor
   */
  public function assertExceptionDescriptor()
  {
    return $this->exception;
  }
```

```
/**
 * assertExceptionCallback
 *
 * Set/get exception callback function
 * @param string $callback = name of the callback function
 */
public function assertExceptionCallback($callback=null)
{
      if ($callback !== null)
      {
            if (method_exists($this, $callback))
            {
                  $this->exceptionCallback = $callback;
            }
      }

      return $this->exceptionCallback;
}

/**
  * assertLogMessage
  *
  * Output a message + newline to the current output device
  * @param string $message = message to output
  * @param integer $level = (optional) log level ('debug', 'error',
  *                                               <user defined>) DEFAULT: debug
  */
public function assertLogMessage($message, $level='debug')
{
      if ($this->verbose)
      {
            Log::message(sprintf("Test #%u, Subtest #%u - %s",
                                 $this->properties->Run_TestNumber,
                                 $this->testNumber,
                                 $message),
                           array("Log_Level"      => $level,
                                 "Log_Program"    => $this->testName,
                                 "Log_Method"     => substr(__METHOD__,
                                                      strpos(__METHOD__, "::") + 2),
                                 "Log_Class"      => $this->getEldestParent($this),
                                 "Log_SkipLevels" => 7));
      }
}
```

```
/**
 * assertStartLogger
 *
 * Create a test-relative log file and start the logger
 * @param string $logName = (optional) internal logger name, default = 'testlogger'
 * @param string $format = (optional) output format name, null to not assign
 * @return boolean $result = true if successful, false if not
 */
public function assertStartLogger($logName='testlogger', $format=null)
{
      if ($this->assertLogger && ($this->assertLogger == $logName))
      {
            $this->assertStopLogger($logName);
      }

      $this->loggerProperties = new \Library\Properties
                                    ($this->properties->setup->getLogDefaults());

      $this->loggerProperties->Log_Name      = $logName;
      $this->loggerProperties->Log_Adapter   = 'fileio';
      $this->loggerProperties->Log_Level     = 'debug';

      if ($format !== null)
      {
            $this->loggerProperties->Log_Format = $format;
      }

      $logFileName = str_replace(array('\\', '_'), '-', $this->testName);
      if (substr($logFileName, 0, 1) == '-')
      {
            $logFileName = substr($logFileName, 1);
      }

        $logFileName = sprintf("% 3u-%s", $this->properties->Run_TestNumber,
                                          $logFileName);

        $this->loggerProperties->Log_FileDestination
              = dirname($this->properties->Log_FileDestination) .
                        DIRECTORY_SEPARATOR . $logFileName;
        $this->loggerProperties->Log_FileAdapter = 'fileobject';
        $this->loggerProperties->Log_FileMode    = 'w';

        try
        {
              Log::startLog($this->loggerProperties);
        }
        catch(\Library\Log\Exception $exception)
        {
              Log::message(sprintf('Unable to open disk file log: %s',
                                   $exception->getMessage()), 'error');
              Log::message('Logging to disk file has been disabled', 'error');
              return false;
        }

        $this->assertLogger = $logName;

        $this->assertLogMessage(sprintf('Test Program: %s', $this->testName));
        if (CliParameters::parameterCount() == 0)
        {
              $this->assertLogMessage('    NO CLI Parameters');
        }
```

```
        else
        {
                $this->assertLogMessage
                (\Library\PrintU\FormatArray::format(CliParameters::parameters(),
                 'CLI Parameters', false, false, false));
        }

        $this->properties->Test_Start = date('Y-m-d H:i:s') .
                                            substr((string)microtime(), 1, 6);
        $this->properties->Test_End   = null;

        $this->assertLogMessage('Start-of-test @ ' . $this->properties->Test_Start);
        return true;
}

/**
 * assertStopLogger
 *
 * Stop the current logger, if it is running
 * @param string $logName = (optional) internal log name to stop,
 *                                     null to stop current logger
 */
public function assertStopLogger($logName=null)
{
        $this->testNumber++;

        $this->properties->Test_End = date('Y-m-d H:i:s') .
                                          substr((string)microtime(), 1, 6);

        $this->assertLogMessage('End-of-test @ ' . $this->properties->Test_End);

        if ($logName == null)
        {
                $logName = $this->assertLogger;
        }

        Log::stopLog($logName);

        if ($logName == $this->assertLogger)
        {
                $this->assertLogger = null;
        }
}
```

```
    /**
     * getEldestParent
     *
     * Get the top parent in the heirarchy
     * @param mixed $object = class name or object to get eldest parent for
     * @return string $class = eldest parent, false if no parent
     * @throws \Library\Testing\Exception
     */
    public function getEldestParent($object)
    {
      if (is_object($object))
      {
            $class = get_class($object);
      }
      elseif (is_string($object))
      {
            $class = $object;
      }
      else
      {
            throw new Exception('StringOrObjectExpected');
      }

      while($parent = get_parent_class($class))
      {
            $class = $parent;
      }

      return $class;
    }

}
```