Testing\UtilityMethods

The ***Testing\UtilityMethods*** class extends the **Testing\Base** class to provide additional utility methods for use with test classes.

Design criteria and considerations

The ***Testing\UtilityMethods*** class

1. extends the **Testing\Base** class to provide additional utility methods for use in **PHPProjectLibrary** class testing;
2. uses the ***Library\Testing*** namespace;
3. uses the ***Library\PrintU\FormatArray*** class to format array output;
4. uses the Library\Error class to display error codes.

Class Properties

The ***Testing\UtilityMethods*** class defines the following properties (variables):

Method	Description
$utility_os	Operation system name.

Inherited Properties

The ***Testing\UtilityMethods*** class inherits 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.

Inherited Static Properties

The ***Testing\UtilityMethods*** class inherits 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\UtilityMethods*** class contains the following methods:

Method	Description
__construct	Class constructor
__destruct	Class destructor
assertionTests	Execute assertion test steps.
a_printArray	Print the array contents.
a_compareArray	Compare an array with ***$a_localArray***.
a_identicalArrays	Compare fields in 2 arrays.
a_compareExpectedType	Compare data with expected value and type.
a_compareExpected	Compare data with expected value.
a_compareType	Compare 2 parameters according to their types.
a_showData	Output the value of a variable to the logging device(s).
a_exceptionCaughtTrue	Check the ***$exceptionCaught*** class property for ***true*** status.
a_exceptionCaughtFalse	Check the ***$exceptionCaught*** class property for ***false*** status.
a_printException	Output the exception to the logging device(s).
a_absoluteFileName	Takes a workspace relative file name and returns an absolute path to the file
a_outputAndDie	Output a message to the logging device(s) and terminate the **phpTest** application.

Inherited Methods

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

Method	Description
__construct	Class constructor
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.

Method	Description
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 ***$exceptionCaught*** class property.
labelBlock	Output a separator block containing a label.
labelBlockFlag	Get/set the label block flag.
assertCallback	Assert test callback function.
properties	Get/set the properties object.
verbose	Get/set the verbose property.
errorsOnly	Display only errors flag, used in **assertTest** to limit output.
testNumber	Get/set the sub-test number.
testName	Get/set the test name.
assertEventArray	Get/set the event array.
assertLogMessage	Output a formatted message to the current output device(s).
assertFailures	Get/set the ***$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. Call the parent class constructor;
2. get the operating system name from the PHP **php_uname** function;
3. convert the operating system name to lower case and assign the result to the ***utility_os*** class property;
4. exit.

Implementation

```
/**
 * __construct
 *
 * Class constructor
 */
public function __construct()
{
      parent::__construct();
      $this->utility_os = strtolower(php_uname('s'));
}
```

Narrative

The PHP **parent** scope resolution operator is used to call the base class constructor.

The PHP **php_uname** function is used to get a copy of the operating system name. The result is passed to the PHP **strtolower** function to convert the operating system name to lower case characters and assigned to the ***$utility_os*** class property.

__destruct

The class destructor.

Algorithm

1. Call the parent class destructor;
2. exit.

Implementation

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

Narrative

The PHP parent scope resolution operator is used to call the base class destructor.

assertionTests

Execute assertion test steps.

Algorithm

1. Accept ***logger*** as the name of the logger to use;
2. accept ***format*** as the output format name;
3. assign the ***eolSequence*** property from 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. exit.

Implementation

```
/**
 * assertionTests
 *
 * run the current assertion test steps
 * @parm string $logger = (optional) name of the logger to use, null for none
 * @param string $format = (optional) output format name, null to not assign
 */
public function assertionTests($logger=null, $format=null)
{
	$this->eolSequence = $this->properties->eolSequence;

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

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

Narrative

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

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

The ***$eolSequence*** class property is set to the value of the ***$eolSequence*** property of the ***$properties*** class property.

If the $logger parameter is not null, the $logger and $format parameters are passed to the assertStartLogger class method.

The **assertFailures** class method is passed a **false** value to inhibit automatic output of PHP **assert** failure messages.

a_printArray

Print the array contents.

Algorithm

1. Accept ***array*** as the array to print and assign it to the ***a_testArray*** property;
2. accept ***label*** as the optional array label to print and assign it to the ***a_label*** property;
3. accept ***sort*** as the sort fields if true and assign it to the ***a_sort*** class property;
4. accept ***sortValues*** as the optional sort-by-value flag and assign it to the ***a_sortValues*** property;
5. accept ***recurse*** as the optional recursion flag and assign it to the ***a_recurse*** property;
6. create a command string containing a call to the **format** method of the **Library\PrintU\FormatArray** class, passing the ***a_testArray***, ***a_label***, ***a_sort***, ***a_sortValue*** and ***a_recurse*** class properties, and assign the result to the ***a_buffer*** class property;
7. assign the command string to ***assertion***;
8. pass ***assertion***, and a message to display with the assertion in it, to the **assertTrue** class method;
9. if the result is true, goto step **11**;
10. pass the ***a_buffer*** class property to the **a_outputAndDie** class;
11. pass the ***a_buffer*** class property to the **assertLogMessage** class method;
12. exit.

Implementation

```
/**
 * a_printArray
 *
 * Print the array contents
 * @param array $array = array to print
 * @param string $label = (optional) array label to print, null for none
 * @param boolean $sort = (optional) sort fields if true (default = false)
 * @param boolean $sortValues = (optional) sort values if true (default = false)
 * @param boolean $recurse = (optional) recursion allowed if true
 */
public function a_printArray($array, $label=null,
                             $sort=false, $sortValues = false, $recurse=false)
{
      $this->labelBlock('Print array.', 40, '*');

      $this->a_testArray = $array;
      $this->a_label = $label;
      $this->a_sort = $sort;
      $this->a_sortValues = $sortValues;
      $this->a_recurse = $recurse;

      $assertion = '$this->a_buffer =
                        \Library\PrintU\FormatArray::format($this->a_testArray,
                                                            $this->a_label,
                                                            $this->a_sort,
                                                            $this->a_sortValues,
                                                            $this->a_recurse);';
```

```
        if (! $this->assertTrue($assertion,
                                sprintf('PrintArray - Asserting: %s', $assertion)))
        {
              $this->a_outputAndDie($this->a_buffer);
        }

        $this->assertLogMessage($this->a_buffer);
  }
```

Narrative

The **a_printArray** class method expects a single (1) mandatory parameter:

Parameter	Description
$array	The array to print.

And four (4) optional parameters:

Parameter	Description
$label	An array label to print.
$sort	Sort array in ascending order, if true.
$sortValues	Sort values is ascending order, if true.
$recurse	Recursion allowed, if true.

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

The ***$array*** parameter is assigned to the ***$a_testArray*** class property; the ***$label*** parameter is assigned to the ***$a_label*** class property; the ***$sort*** parameter is assigned to the ***$a_sort*** class property; and the ***$resurse*** parameter is assigned to the ***$a_recurse*** class property.

An assertion command is created as follows:

> The ***$a_testarray***, ***$a_label***, ***$a_sort*** and ***$a_recurse*** class properties are passed to the **format** class method of the **Library\PrintU\FormatArray** class. The result is assigned to the ***$a_buffer*** class property.

The assertion command is assigned to the ***$assertion*** method variable.

The ***$assertion*** method variable and an identifying message is passed to the **assertTrue** class method. If the value returned is false, the ***$a_buffer*** class property is passed to the **a_outputAndDie** class method.

The ***$a_buffer*** class property is passed to the **assertLogMessage** class method.

a_compareArray

Compare an array with the ***$a_localArray*** class property.

Algorithm

1. Assign true to the ***a_type*** class property;
2. accept **array** as the array to compare with the ***a_localArray*** and assign it to the ***a_array*** class property;
3. accept ***type*** as the optional type of comparison and assign it to the ***a_type*** class property, if it is not null;
4. accept ***localArray*** as the optional array to compare the ***a_array*** class property to and assign it to the ***a_localArray*** class property, if it is not null;
5. create a command string containing a call to the ***a_identicalArrays*** and passing the ***a_localArray*** and ***a_array*** class properties;
6. assign the command string to ***assertion***;
7. if ***a_type*** is false, goto step **14**;
8. pass ***assertion***, and an assertion message to be printed, to the **assertTrue** class method;
9. if the result is true, goto step **16**;
10. pass ***a_localArray***, and the string '**localArray**' as a label, to the ***a_printArray*** class method;
11. pass ***a_array***, and the string '**array**' as a label, to the ***a_printArray*** class method;
12. call the ***a_outputAndDie*** class method;
13. goto step **16**;
14. pass ***assertion***, and an assertion message to be printed to the **assertFalse** class method;
15. if the result is false, goto step **10**;
16. exit.

Implementation

```
/**
 * a_compareArray
 *
 * Compare with local array
 * @param array $array = array to compare with local array
 * @param boolean $type = (optional) type of comparison (true or false),
 *                        default = true
 * @param array $localArray = (optional) local array to compare $array to
 */
public function a_compareArray($array, $type=true, $localArray=null)
{
      $this->labelBlock('Compare Array.', 40, '*');

      $this->a_array = $array;

      if ($localArray)
      {
            $this->a_localArray = $localArray;
      }

      $assertion = '$this->a_identicalArrays($this->a_localArray, $this->a_array);';
```

```
        if ($this->a_type = $type)
        {
              if (! $this->assertTrue($assertion,
                                      sprintf('Compare arrays TRUE - Asserting: %s',
                                              $assertion)))
              {
                    $this->a_printArray($this->a_localArray, 'localArray');
                    $this->a_printArray($this->a_array, 'array');
                    $this->a_outputAndDie();
              }
        }
        else
        {
              if (! $this->assertFalse($assertion,
                                       sprintf('Compare arrays FALSE - Asserting: %s',
                                               $assertion)))
              {
                    $this->a_printArray($this->a_localArray, 'localArray');
                    $this->a_printArray($this->a_array, 'array');
                    $this->a_outputAndDie();
              }
        }
  }
```

Narrative

The **a_printArray** class method expects a single (1) mandatory parameter:

Parameter	Description
$array	The array to compare to ***$a_localArray***.

And two (2) optional parameters:

Parameter	Description
$type	The type of comparison (true or false). Default is **true**.
$localArray	The array to compare ***$array*** to. ***$a_localArray*** is used if ***$localArray*** is null.

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

The $array parameter is assigned to the ***$a_array*** class property.

If the ***$localArray*** is not null, the ***$localArray*** parameter is assigned to the ***$a_localArray*** class property.

An assertion command is created as follows:

The ***$a_array*** and ***$a_localArray*** class properties are passed to the **a_identicalArrays** class method.

The assertion command is assigned to the ***$assertion*** method variable.

The ***$type*** parameter is assigned to the ***$a_type*** class property.

If the ***$a_type*** class property is **true**,

The ***$assertion*** method variable and an identifying message is passed to the **assertTrue** class method.

Otherwise,

The ***$assertion*** method variable and an identifying message is passed to the **assertFalse** class method.

If the value returned from the assertion test is false the ***$a_localArray*** class property is passed to the **a_printArray** class method; the ***$a_array*** class property is passed to the **a_printArray** class method; and the **a_outputAndDie** class method is called to exit the **phpTest** application.

a_identicalArrays

Compare fields in ***$compareArray*** with fields in ***$array***.

Algorithm

1. Accept ***array*** as array to compare to;
2. accept ***compareArray*** as array to compare from;
3. if ***array*** is an array, goto **5**;
4. exit with a **false** result;
5. if ***compareArray*** is not an array, goto step **4**;
6. if the size of ***array*** is not equal to the size of ***compareArray***, goto step 4;
7. reset array pointer to the start of ***array***;
8. extract the current element of ***array*** to ***field***;
9. if the array pointer does not exist as an index to ***compareArray***, goto step **4**;
10. assign the element at array pointer in ***compareArray*** to ***from***;
11. if ***field*** is not an object reference, goto step **15**;
12. if ***from*** is not an object reference, goto step 4;
13. assign the name of the object in ***from*** to ***from***;
14. assign the name of the object in ***field*** to ***field***;
15. if ***field*** is not equal to ***from***, goto step **4**;
16. if array pointer is the last element in ***array***, goto step **19**;
17. advance array pointer to the next element in ***array***;
18. goto step **8**;
19. exit with a **true** result.

Implementation

```
/**
 * a_identicalArrays
 *
 * Compare fields in $compareArray with fields in $array -
 *         if the two arrays are identical, return true
 * If a class object is in the array, the names of the classes will be compared
 * NOTE: This may be slow, but it is more accurate and less prone to errors
 * @param array $array = array to compare to
 * @param array $compareArray = array to compare from
 * @return boolean $result = true if identical, false if not
 */
public function a_identicalArrays($array, $compareArray)
{
      if ((! is_array($array)) || (! is_array($compareArray)) ||
          (count($array) !== count($compareArray)))
      {
            return false;
      }

      foreach($array as $index => $field)
      {
            if (! array_key_exists($index, $compareArray))
            {
                  return false;
            }

            $from = $compareArray[$index];
```

```
                if (is_object($field))
                {
                        if (! is_object($from))
                        {
                                return false;
                        }

                        $from = get_class($from);
                        $field = get_class($field);
                }

                if ($from !== $field)
                {
                        return false;
                }
        }

        return true;
}
```

Narrative

The **a_identicalArrays** class method expects two (2) mandatory parameters:

Parameter	Description
$array	The array to compare to.
$compareArray	The array to compare from.

The ***$array*** parameter is passed to the PHP **is_array** function. If the result is **false**, a **false** value is returned.

The ***$compareArray*** parameter is passed to the PHP **is_array** function. If the result is **false**, a **false** value is returned.

The ***$array*** and ***$compareArray*** parameters are both passed to the PHP **count** function. If the results are not equal, a **false** value is returned.

Iterate through each element of ***$array*** method array, extracting the array's index to the ***$index*** method variable and the array's value to the ***$field*** method variable. For each ***$index*** method variable,

pass the ***$index*** method variable and the ***$compareArray*** parameter to the PHP **array_key_exists** function. If the result is **false**, return a **false** value;

assign the value stored at the ***$index*** method value in the ***$compareArray*** parameter to the ***$from*** method value;

the ***$field*** method variable is passed to the PHP **is_object** function. If the result is **true**,

the ***$from*** method variable is passed to the PHP **is_object** function. If the

result is **false**, a **false** value is returned;

the ***$from*** method variable is passed to the PHP **get_class** function and the result is assigned to the ***$from*** method variable;

the ***$field*** method variable is passed to the PHP **get_class** function and the result is assigned to the ***$field*** method variable;

if ***$field*** is not equal to ***$from***, a **false** value is returned.

A **true** value is returned.

a_compareExpectedType

Compare the value in ***$a_data*** with the expected result and expected type.

Algorithm

1. Accept ***type*** as the type of comparison;
2. accept ***expected*** as the expected result;
3. accept ***data*** as the data to compare to;
4. if ***expected*** is null, goto step **13**;
5. if ***data*** is null, goto step 7;
6. assign ***data*** to the ***a_data*** class property;
7. if the ***a_data*** class property is not an array, goto step **12**;
8. if ***expected*** is not an array, goto step **12**;
9. assign the ***a_data*** class property to the ***a_localArray*** class property;
10. pass ***expected*** and ***type*** to the **a_compareArray** class method;
11. goto step **13**;
12. pass ***expected*** and ***type*** to the **a_compareExpected** class method;
13. exit.

Implementation

```
/**
 * a_compareExpectedType
 *
 * Compare the value in a_data with the expected result, and expected type
 * @param boolean $type = type of comparison (true or false)
 * @param mixed $expected = expected result
 * @param mixed $data = (optional) data to compare to, a_data is used if null
 */
public function a_compareExpectedType($type, $expected, $data=null)
{
	$this->labelBlock('Compare expected type.', 40, '*');

	if ($expected === null)
	{
		return;
	}

	if ($data !== null)
	{
		$this->a_data = $data;
	}

	if (is_array($this->a_data) && is_array($expected))
	{
		$this->a_localArray = $this->a_data;
		$this->a_compareArray($expected, $type);
	}
	else
	{
		$this->a_compareExpected($expected, $type);
	}
}
```

Narrative

The **a_compareExpectedType** class method expects a two (2) mandatory parameter:

Parameter	Description
$type	Type of comparison (true or false).
$expected	Expected result.

And a single (1) optional parameter

Parameter	Description
$data	**Data to compare to, $a_data if null**.

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

If the ***$expected*** parameter is null, return.

If the ***$data*** parameter is not null, the ***$data*** parameter is assigned to the ***$a_data*** class property.

The ***$a_data*** class property and the ***$expected*** parameter is passed to the PHP **is_array** function, one at a time. If either result is false, the ***$expected*** parameter and the ***$type*** parameter is passed to the **a_compareExpected** class method. Otherwise, the ***$a_data*** class property is assigned to the ***$a_localArray*** class property and the ***$expected*** and ***$type*** parameters are passed to the **a_compareArray** class method.

a_compareExpected

Compare the value in the ***$a_data*** class property with the expected value.

Algorithm

1. Accept ***expected*** as the expected result;
2. accept ***type*** as the optional type of comparison;
3. if ***type*** is null, set ***type*** to true;
4. if the ***verbose*** class property is less than or equal to 1, goto step **8**;
5. pass ***expected***, and a string containing '**expected**', to the **a_showData** class method;
6. pass ***type***, and a string containing '**type**', to the **a_showData** class method;
7. pass the ***a_data*** class property, and a string containing '**a_data**', to the **a_showData** class method;
8. assign ***type*** to the ***a_expected*** class property;
9. if the first character in ***a_expected*** is not "\\" goto step **13**;
10. if the first character in the ***a_data*** class property is "\\" goto step **13**;
11. delete the first character from ***a_expected***;
12. goto step **16**;
13. if the first character in the ***a_data*** class property is not "\\" goto step **16**;
14. if the first character in the ***a_expected*** class property is "\\" goto step **16**;
15. delete the first character in the ***a_data*** class property;
16. create a command string containing a call to the ***a_compareType*** and passing the ***a_data*** and ***a_expected*** class properties;
17. assign the command string to ***assertion***;
18. pass ***assertion***, and an assertion message to be printed, to the **assertTrue** class method;
19. if the result is true, goto step **24**;
20. call the ***a_outputAndDie*** class method;
21. goto step **24**;
22. pass ***assertion***, and an assertion message to be printed to the **assertFalse** class method;
23. if the result is false, goto step **20**;
24. exit.

Implementation

```
/**
 * a_compareExpected
 *
 * Compare the value in a_data with the expected result
 * @param mixed $expected = expected result
 * @param boolean $type = type of comparison (true or false)
 */
public function a_compareExpected($expected, $type=true)
{
	$this->labelBlock('Compare expected.', 40, '*');

	if ($this->verbose > 1)
	{
		$this->a_showData($expected, 'expected');
		$this->a_showData($type, 'type');
```

```
            $this->a_showData($this->a_data, 'a_data');
        }

        $this->a_expected = $expected;
        if (substr($this->a_expected, 0, 1) == '\\' &&
            substr($this->a_data, 0, 1) !== '\\')
        {
            $this->a_expected = substr($this->a_expected, 1);
        }
        elseif (substr($this->a_data, 0, 1) == '\\' &&
                substr($this->a_expected, 0, 1)!== '\\')
        {
            $this->a_data = substr($this->a_data, 1);
        }

        $assertion = '$this->a_compareType($this->a_data, $this->a_expected);';
        if ($type)
        {
            if (! $this->assertTrue($assertion,
                                    sprintf('Checking result - Asserting: %s',
                                            $assertion)))
            {
                $this->a_outputAndDie();
            }
        }
        else
        {
            if (! $this->assertFalse($assertion,
                                     sprintf('Checking result - Asserting: %s',
                                             $assertion)))
            {
                $this->a_outputAndDie();
            }
        }
    }
```

Narrative

The **a_compareExpected** class method expects a two (2) mandatory parameters:

Parameter	Description
$expected	Expected result.
$type	Type of comparison (true or false).

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

If the ***$verbose*** class property is greater than 1,

the ***$expected*** paramerter and the string '**expected**' is passed to the **a_showData** class method for formatting and output to the logging device(s);

the ***$type*** parameter and the string '**type**' is passed to the **a_showData** class method for formatting and output to the logging device(s);

the ***$a_data*** class property and the string '**a_data**' is passed to the **a_showData**

class method for formatting and output to the logging device(s).

The ***$expected*** parameter is assigned to the ***$a_expected*** class property.

The ***$a_expected*** class property is passed to the PHP **substr** function to isolate the first character. If the first character is equal to "\\",

the ***$a_data*** class property is passed to the PHP **substr** function to isolate the first character. If the first character is not equal to "\\", the ***$a_expected*** class property is passed to the PHP **substr** function to eliminate the first character of the string and the result is assigned to the ***$a_expected*** class property;

Otherwise, the ***$a_data*** class property is passed to the PHP **substr** function to isolate the first character. If the first character is equal to "\\",

the ***$a_expected*** class property is passed to the PHP **substr** function to isolate the first character. If the first character is not equal to "\\", the ***$a_data*** class property is passed to the PHP **substr** function to eliminate the first character of the string and the result is assigned to the ***$a_data*** class property.

An assertion command is created as follows:

The ***$a_data*** and ***$a_expected*** class properties are passed to the **a_compareType** class method.

The assertion command is assigned to the ***$assertion*** method variable.

If the ***$type*** parameter is **true**,

The ***$assertion*** method variable and an identifying message is passed to the **assertTrue** class method.

Otherwise,

The ***$assertion*** method variable and an identifying message is passed to the **assertFalse** class method.

If the value returned from the assertion test is false, the **a_outputAndDie** class method is called.

a_expectedType

Compare the 2 parameters according to their types.

Algorithm

1. Accept ***data*** as the first parameter to compare;
2. accept ***compareTo*** as the second parameter to compare;
3. assign the type of ***data*** to ***dataType***;
4. if ***dataType*** is not equal to the type of compareTo goto step **19**;
5. if ***dataType*** is equal to '**string**' goto step **10**;
6. if ***dataType*** is equal to '**integer**' goto step **10**;
7. if ***dataType*** is equal to '**double**' goto step **10**;
8. if ***dataType*** is equal to '**boolean**' goto step **10**;
9. if ***dataType*** is not equal to '**NULL**' goto step **12**;
10. if ***data*** is not equal to ***compareTo*** goto step **19**;
11. exit with a **true** result;
12. if ***dataType*** is not equal to '**array**' goto step **14**;
13. pass ***data*** and ***compareTo*** to the ***a_identicalArrays*** class method and exit with the returned result;
14. if ***dataType*** is not equal '**object**' goto step **17**;
15. if the ***data*** class name is equal to the ***compareTo*** class name, goto step **11**;
16. goto step **19**;
17. if ***dataType*** is not equal to '**resource**' goto step **19**;
18. if the ***data*** resource type is equal to the ***compareTo*** resource type, goto step **11;**
19. exit with a **false** result.

Implementation

```
/**
 * a_compareType
 *
 * Compare the two parameters according to their types.
 * @param mixed $data = first parameter to compare.
 * @param mixed $compareTo = second parameter to compare.
 * @return boolean $result = true if equal, false if not.
 */
public function a_compareType($data, $compareTo)
{
	$dataType = gettype($data);
	if ($dataType === gettype($compareTo))
	{
		switch($dataType)
		{
			case 'string':
			case 'integer':
			case 'double':
			case 'boolean':
			case 'NULL':
				if ($data === $compareTo)
				{
					return true;
				}
```

```
                        break;

                case 'array':
                        return a_identicalArrays($data, $compareTo);

                case 'object':
                        if (get_class($data) === get_class($compareTo))
                        {
                                return true;
                        }

                        break;

                case 'resource':
                        if (@get_resource_type($data) ===
                            @get_resource_type($compareTo))
                        {
                                return true;
                        }

                        break;

                default:
                        break;
                }
        }

        return false;
}
```

Narrative

The **a_compareType** class method expects a two (2) mandatory parameters:

Parameter	Description
$data	First parameter to compare.
$compareTo	Second parameter to compare.

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

The ***$data*** parameter is passed to the PHP **gettype** function and the result is assigned to the ***$dataType*** method variable.

The ***$compareTo*** parameter is passed to the PHP **gettype** function. If the result is not equal to the ***$dataType*** method variable, a **false** value is returned.

A PHP **switch** statement is used to perform the following comparisons:

If the ***$dataType*** method variable is equal to '**string**', '**integer**', '**double**', '**boolean**' or **NULL**,

if the ***$data*** parameter is equal to the ***$compareTo*** parameter, a **true** value

is returned;

Otherwise, if the ***$dataType*** method variable is equal to '**array**', the ***$data*** and ***$compareTo*** parameters are passed to the **a_identicalArrays** class method and the result is returned;

Otherwise, if the ***$dataType*** method variable is equal to '**object**',

the ***$data*** and ***$compareTo*** parameters are passed to the PHP **get_class** function, one at a time; if both results are equal, a **true** value is returned;

Otherwise, if the ***$dataType*** method variable is equal to '**resource**',

the ***$data*** and ***$compareTo*** parameters are passed to the PHP **get_resource_type** function, one at a time; if both results are equal, a **true** value is returned.

A **false** value is returned.

a_showData

Output the value of the (optionally) named variable.

Algorithm

1. Accept ***value*** as the value to be output;
2. accept ***name*** as the optional name of the variable;
3. assign the string '**VALUE**' to ***buffer***;
4. if ***name*** is not null append the string ' **of** ' and ***name*** to ***buffer***;
5. append the data type of ***value*** to ***type***;
6. append the string '**: (**', ***type*** and '**)**' to ***buffer***;
7. if ***type*** is not equal to '**string**' goto step **10**;
8. cast ***value*** as a string and append the result to ***buffer***;
9. goto step **32**;
10. if ***type*** is not equal to '**boolean**' goto step **13**;
11. append ***value*** as the boolean value to ***buffer***;
12. goto step **32**;
13. if ***type*** is not equal to '**integer**' goto step **16**;
14. append **value** as the integer value to ***buffer***;
15. goto step **32**;
16. if ***type*** is not equal to '**double**' or '**float**' goto step **19**;
17. append ***value*** as the double value to ***buffer***;
18. goto step **32**;
19. if ***type*** is not equal '**array**' goto step **22**;
20. append the result of passing ***value*** to the **format** method of the **Library\PrintU\FormatArray** class;
21. goto step **32**;
22. if ***type*** is not equal to '**object**' goto step **25**;
23. append the object name of ***value*** to ***buffer***;
24. goto step **32**;
25. if ***type*** is not equal to '**resource**' goto step **28**;
26. append the resource type of ***value*** to ***buffer***;
27. goto step **32**;
28. if ***type*** is not equal to '**null**' goto step **31**;
29. append the the string '**null**' to ***buffer***;
30. goto step **32**;
31. append the value of ***value*** to ***buffer***;
32. pass ***buffer*** to the **assertLogMessage** class method;
33. exit.

Implementation

```
/**
 * a_showData
 *
 * Output the value of the (optionally) named variable
 * @param mixed $value = value to output
 * @param string $name = (optional) name of the variable
 */
public function a_showData($value, $name=null)
{
```

```
        $buffer = 'VALUE';

        if ($name)
        {
              $buffer .= sprintf(' of %s', $name);
        }

        $type = gettype($value);
        $buffer .= sprintf(': (%s)', $type);

        switch($type)
        {
              case 'string':
                    $buffer .= (string)$value;
                    break;

              case 'boolean':
                    $buffer .= sprintf('%b', $value);
                    break;

              case 'integer':
                    $buffer .= sprintf('%d', $value);
                    break;

              case 'double':
              case 'float':
                    $buffer .= sprintf('%f', $value);
                    break;

              case 'array':
                    $buffer .= \Library\PrintU\FormatArray::format($value);
                    break;

              case 'object':
                    $buffer .= get_class($value);
                    break;

              case 'resource':
                    $buffer .= get_resource_type($value);
                    break;

              case 'NULL':
                    $buffer .= 'null';
                    break;

              case 'unknown type':
              default:
                    $buffer .= sprintf('%s', $value);
                    break;
        }

        $this->assertLogMessage($buffer);
  }
```

Narrative

The **a_compareType** class method expects a single (1) mandatory parameter:

Parameter	Description

$value	Value to be output.

And a single (1) optional parameter:

Parameter	Description
$name	Name of the variable.

Set the ***$buffer*** method variable to a string of '**VALUE**'.

If the ***$name*** parameter is not null, append a string of ' **of** ' followed by the name to the ***$buffer*** method variable.

The string '**:** ' is appended to the ***$buffer*** method variable.

The ***$value*** parameter is passed to the PHP **gettype** function and the result is assigned to the ***$type*** method variable.

The ***$type*** method variable is converted to a string within parenthesis and appended to the ***$buffer*** method variable.

A PHP **switch** statement is used to perform the following comparisons:

if the ***$type*** method variable is '**string**' the $value parameter is cast to a string and appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**boolean**' the ***$value*** parameter is passed to the PHP **sprintf** function to convert it to a printable boolean string and the result is appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**integer**' the ***$value*** parameter is passed to the PHP **sprintf** function to convert it to a printable integer string and the result is appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**double**' or '**float**' the ***$value*** parameter is passed to the PHP **sprintf** function to convert it to a printable float string and the result is appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**array**' the ***$value*** parameter is passed to the **format** method of the **Library\PrintU\FormatArray** class to convert it to a printable array and the result appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**object**' the ***$value*** parameter is passed to the PHP **get_class** function and the result is appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**resource**' the ***$value*** parameter is passed to the PHP **get_resource_type** function and the result is appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is **NULL** the string '**null**' is appended to the ***$buffer*** method variable;

otherwise, if the ***$type*** method variable is '**unknown type**' the ***$value*** parameter is passed to the PHP **sprintf** function to convert it to a string and the result is appended to the ***$buffer*** method variable;

The ***$buffer*** method variable is passed to the **assertLogMessage** class method for output to the logging device(s).

a_exceptionCaughtTrue

Check the ***$exceptionCaught*** class property for **true** status.

Algorithm

1. Accept ***exit*** as an optional variable indicating report on assert failure, if true;
2. create a command string containing a call to the **exceptionCaught** class method;
3. assign the command string to ***assertion***;
4. pass ***assertion***, and an assertion message to be printed, to the **assertTrue** class method;
5. if the result is **true**, goto step **10**;
6. call the **a_printException** class method;
7. if exit is **false**, goto step **10**;
8. call the ***a_outputAndDie*** class method;
9. exit with a **false** result;
10. exit with a **true** result.

Implementation

```
/**
 * a_exceptionCaughtTrue
 *
 * check the exception caught for true status
 * @param boolean $exit = true to report and exit on failure, false to return
 */
public function a_exceptionCaughtTrue($exit=true)
{
	$this->labelBlock('Exception caught true.', 40, '*');

	$assertion = '$this->exceptionCaught()';
	if (! $this->assertTrue($assertion, sprintf('Asserting: %s', $assertion)))
	{
		$this->a_printException();
		if ($exit)
		{
			$this->a_outputAndDie();
		}
	}
}
```

Narrative

The **a_compareType** class method expects a single (1) optional parameter:

Parameter	Description
$exit	True (default) to report and exit on failure, false to report and return.

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

An assertion command is created as follows:

The ***$exceptionCaught*** class method is called.

The assertion command is assigned to the ***$assertion*** method variable.

The ***$assertion*** method variable and an identifying message is passed to the **assertTrue** class method. If the result is **false**,

the **a_printException** class method is called;

if the ***$exit*** parameter is **true**, the **a_outputAndDie** class method is called.

a_exceptionCaughtFalse

Check the ***$exceptionCaught*** class property for **false** status.

Algorithm

1. Accept ***exit*** as an optional variable indicating report on assert failure, if true;
2. create a command string containing a call to the **exceptionCaught** class method;
3. assign the command string to ***assertion***;
4. pass ***assertion***, and an assertion message to be printed, to the **assertFalse** class method;
5. if the result is **true**, goto step **10;**
6. call the **a_printException** class method;
7. if exit is **false**, goto step **10**;
8. call the ***a_outputAndDie*** class method;
9. exit with a **false** result;
10. exit with a **true** result.

Implementation

```
/**
 * a_exceptionCaughtTrue
 *
 * check the exception caught for true status
 * @param boolean $exit = true to report and exit on failure, false to return
 */
public function a_exceptionCaughtTrue($exit=true)
{
	$this->labelBlock('Exception caught true.', 40, '*');

	$assertion = '$this->exceptionCaught()';
	if (! $this->assertFalse($assertion, sprintf('Falsely Asserting: %s',
							$assertion)))
	{
		$this->a_printException();
		if ($exit)
		{
			$this->a_outputAndDie();
		}
	}
}
```

The **a_compareType** class method expects a single (1) optional parameter:

Parameter	Description
$exit	True (default) to report and exit on failure, false to report and return.

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

An assertion command is created as follows:

The ***$exceptionCaught*** class method is called.

The assertion command is assigned to the ***$assertion*** method variable.

The ***$assertion*** method variable and an identifying message is passed to the **assertFalse** class method. If the result is **false**,

the **a_printException** class method is called;

if the ***$exit*** parameter is **true**, the **a_outputAndDie** class method is called.

a_printException

Output the excepton to the logging device(s) as a printable string.

Algorithm

1. Cast the ***exception*** class property as a string and pass the result to the **assertLogMessage** class method;
2. exit.

Implementation

```
/**
 * a_printException
 *
 * Output the exception to the log as a printable string
 */
public function a_printException()
{
	$this->assertLogMessage((string)$this->exception);
}
```

Narrative

The ***$exception*** class property is cast as a string and passed to the **assertLogMessage** class method for output to the current logging device(s).

a_absoluteFileName

Takes a workspace relative file name and returns an absolute path to the file

Algorithm

1. Accept ***name*** as the workspace-relative file name;
2. if name is a local stream file, goto step **5**;
3. assign name to the ***a_fileName*** class property;
4. exit with ***a_fileName*** as the result;
5. replace all '/' and '//' characters in name with the PHP **DIRECTORY_SEPARATOR**, and assign the result to the ***a_fileName*** class property;
6. if the first character in ***a_fileName*** class property is equal to the PHP **DIRECTORY_SEPARATOR**, goto step 8;
7. assign the result of concatenating the ***RootPath*** element of the ***properties*** class property, the PHP **DIRECTORY_SEPARATOR** constant, and the ***a_fileName*** class property to the ***a_fileName*** class property;
8. exit with the result equal to the ***a_fileName*** class property.

Implementation

```
/**
 * a_absoluteFileName
 *
 * Takes a workspace relative file name and returns an absolute path to the file
 * @param string $name = workspace relative file name
 * @return string $name = absolute path to the file
 */
public function a_absoluteFileName($name)
{
	if (! stream_is_local($name))
	{
		return $this->a_fileName = $name;
	}

	$this->a_fileName = str_replace(array('/', '\\'), DIRECTORY_SEPARATOR, $name);

	if (substr($this->a_fileName, 0, 1) != DIRECTORY_SEPARATOR)
	{
		$this->a_fileName = sprintf("%s%s%s", $this->properties['RootPath'],
					DIRECTORY_SEPARATOR, $this->a_fileName);
	}

	return $this->a_fileName;
}
```

Narrative

The **a_absoluteFileName** class method expects a single (1) mandatory parameter:

Parameter	Description
$name	Workspace-relative file name.

The ***$name*** parameter is passed to the PHP **stream_is_local** function. If the result is **false**, the ***$name*** parameter is stored in the ***$a_fileName*** class property and the ***$a_fileName*** class property is returned.

An array containing the two strings '/' and '//', the PHP **DIRECTORY_SEPARATOR** string constant and the ***$name*** parameter are passed to the PHP **str_replace** function to replace all occurances of the two strings in the array found in the ***$name*** parameter with the PHP **DIRECTORY_SEPARATOR** string constant. The result is stored in the ***$a_fileName*** class property.

The PHP **substr** function is passed the ***$a_fileName*** class property to isolate the first character. If the first character is not the same as the PHP **DIRECTORY_SEPARATOR**, the '**RootPath**' entry in the ***$properties*** class property, the PHP **DIRECTORY_SEPARATOR**, and the ***$a_fileName*** class property is passed to the PHP **sprintf** function to concate the parameters together, and the result is assigned to the ***$a_fileName*** class property.

The ***$a_fileName*** class property is returned.

a_outputAndDie

Output a message and die.

Algorithm

1. Accept ***message*** as the optional message to output;
2. if ***message*** is null, goto step **5**;
3. pass ***message*** to the **assertLogMessaage** class method;
4. pass ***message*** to the **log** method of the **Logger_Object** in the **properties** class property;
5. throw a **Testing\Exception**, passing ***message*** and the **Library\Error** code corrseponding to the '**PhpTestingFailure**' constant.

Implementation

```
/**
 * a_outputAndDie
 *
 * Output a message and terminate the program
 * @param string $message = (optional) message to output
 */
public function a_outputAndDie($message='')
{
	if ($message != '')
	{
		$this->assertLogMessage($message);
		$this->properties->Logger_Object->log($message, 'error');
	}

	throw new Exception($message, \Library\Error::code('PhpTestingFailure'));
}
```

Narrative

The **a_outputAndDie** class method expects a single (1) optional parameter:

Parameter	Description
$message	Message to output.

If the ***$messaage*** parameter is not empty (or null),

the ***$message*** parameter is passed to the **assertLogMessage** class method;

the ***$message*** parameter and a string containing '**error**' is passed to the **log** method of the **Logger_Object** property of the ***$properties*** class property.

A **Testing\Exception** is thrown containing the ***$message*** parameter and the result of passing the '**PhpTestingFailure**' string to the **code** method of the **Library\Error** class.

Source Listing

```
<?php
namespace Library\Testing;

/*
 *     UtilityMethods 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
*/
/**
 * UtilityMethods
 *
 * Some utility classes for use with Library\Testing\Base -
 *      intended to be extended by test class
 * @author Jay Wheeler
 * @version 1.0
 * @copyright © 2012, 2013 EarthWalk Software.
 * @license Licensed under the Academic Free License version 3.0.
 * @package Library
 * @subpackage Testing
 */
class UtilityMethods extends \Library\Testing\Base
{
       /**
        * utility_os
        *
        * Contains the name of the current operating system, in lower case characters.
        * @var string utility_os
        */
       protected $utility_os;

       /**
        * __construct
        *
        * Class constructor
        */
       public function __construct()
       {
              parent::__construct();
              $this->utility_os = strtolower(php_uname('s'));
       }

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

```
/**
 * assertionTests
 *
 * run the current assertion test steps
 * @parm string $logger = (optional) name of the logger to use, null for none
 * @param string $format = (optional) output format name, null to not assign
 */
public function assertionTests($logger=null, $format=null)
{
	$this->eolSequence = $this->properties->eolSequence;

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

	$this->assertFailures(false);
}

/**
 * a_printArray
 *
 * Print the array contents
 * @param array $array = array to print
 * @param string $label = (optional) array label to print, null for none
 * @param boolean $sort = (optional) sort fields if true (default = false)
 * @param boolean $sortValues = (optional) sort values if true (default = false)
 * @param boolean $recurse = (optional) recursion allowed if true
 */
public function a_printArray($array, $label=null,
                             $sort=false, $sortValues = false, $recurse=false)
{
	$this->labelBlock('Print array.', 40, '*');

	$this->a_testArray = $array;
	$this->a_label = $label;
	$this->a_sort = $sort;
	$this->a_sortValues = $sortValues;
	$this->a_recurse = $recurse;

	$assertion = '$this->a_buffer = ' .
	             '\Library\PrintU\FormatArray::format($this->a_testArray, ' .
	             '$this->a_label, $this->a_sort, $this->a_sortValues, ' .
	             '$this->a_recurse);';
	if (! $this->assertTrue($assertion,
	                        sprintf('PrintArray - Asserting: %s', $assertion)))
	{
		$this->a_outputAndDie($this->a_buffer);
	}

	$this->assertLogMessage($this->a_buffer);
}
```

```
/**
 * a_compareArray
 *
 * Compare with local array
 * @param array $array = array to compare with local array
 * @param boolean $type = (optional) type of comparison (true or false),
 *                        default = true
 * @param array $localArray = (optional) local array to compare $array to
 */
public function a_compareArray($array, $type=true, $localArray=null)
{
	$this->labelBlock('Compare Array.', 40, '*');

	$this->a_array = $array;

	if ($localArray)
	{
		$this->a_localArray = $localArray;
	}

	$assertion = '$this->a_identicalArrays($this->a_localArray, ' .
		 '$this->a_array);';

	if ($this->a_type = $type)
	{
		if (! $this->assertTrue($assertion,
			sprintf('Compare arrays TRUE - Asserting: %s', $assertion)))
		{
			$this->a_printArray($this->a_localArray, 'localArray');
			$this->a_printArray($this->a_array, 'array');
			$this->a_outputAndDie();
		}
	}
	else
	{
		if (! $this->assertFalse($assertion,
			sprintf('Compare arrays FALSE - Asserting: %s', $assertion)))
		{
			$this->a_printArray($this->a_localArray, 'localArray');
			$this->a_printArray($this->a_array, 'array');
			$this->a_outputAndDie();
		}
	}
}
```

```
/**
 * a_identicalArrays
 *
 * Compare fields in $compareArray with fields in $array -
 *           if the two arrays are identical, return true
 * If a class object is in the array, the names of the classes will be compared
 * NOTE: This may be slow, but it is more accurate and less prone to errors
 * @param array $array = array to compare to
 * @param array $compareArray = array to compare from
 * @return boolean $result = true if identical, false if not
 */
public function a_identicalArrays($array, $compareArray)
{
      if ((! is_array($array)) || (! is_array($compareArray)) ||
          (count($array) !== count($compareArray)))
      {
            return false;
      }

      foreach($array as $index => $field)
      {
            if (! array_key_exists($index, $compareArray))
            {
                  return false;
            }

            $from = $compareArray[$index];

            if (is_object($field))
            {
                  if (! is_object($from))
                  {
                        return false;
                  }

                  $from = get_class($from);
                  $field = get_class($field);
            }

            if ($from !== $field)
            {
                  return false;
            }
      }

      return true;
}
```

```
/**
 * a_compareExpectedType
 *
 * Compare the value in a_data with the expected result, and expected type
 * @param boolean $type = type of comparison (true or false)
 * @param mixed $expected = expected result
 * @param mixed $data = (optional) data to compare to, a_data is used if null
 */
public function a_compareExpectedType($type, $expected, $data=null)
{
	$this->labelBlock('Compare expected type.', 40, '*');

	if ($expected === null)
	{
		return;
	}

	if ($data !== null)
	{
		$this->a_data = $data;
	}

	if (is_array($this->a_data) && is_array($expected))
	{
		$this->a_localArray = $this->a_data;
		$this->a_compareArray($expected, $type);
	}
	else
	{
		$this->a_compareExpected($expected, $type);
	}
}
```

```
/**
 * a_compareExpected
 *
 * Compare the value in a_data with the expected result
 * @param mixed $expected = expected result
 * @param boolean $type = type of comparison (true or false)
 */
public function a_compareExpected($expected, $type=true)
{
	$this->labelBlock('Compare expected.', 40, '*');

	if ($this->verbose > 1)
	{
		$this->a_showData($expected, 'expected');
		$this->a_showData($type, 'type');
		$this->a_showData($this->a_data, 'a_data');
	}

	$this->a_expected = $expected;
	if (substr($this->a_expected, 0, 1) == '\\' &&
	    substr($this->a_data, 0, 1) !== '\\')
	{
		$this->a_expected = substr($this->a_expected, 1);
	}
	elseif (substr($this->a_data, 0, 1) == '\\' &&
		 substr($this->a_expected, 0, 1)!== '\\')
	{
		$this->a_data = substr($this->a_data, 1);
	}

	$assertion = '$this->a_compareType($this->a_data, $this->a_expected);';
	if ($type)
	{
		if (! $this->assertTrue($assertion,
				sprintf('Checking result - Asserting: %s', $assertion)))
		{
			$this->a_outputAndDie();
		}
	}
	else
	{
		if (! $this->assertFalse($assertion,
				sprintf('Checking result - Asserting: %s', $assertion)))
		{
			$this->a_outputAndDie();
		}
	}
}
```

```
/**
 * a_compareType
 *
 * Compare the two parameters according to their types.
 * @param mixed $data = first parameter to compare.
 * @param mixed $compareTo = second parameter to compare.
 * @return boolean $result = true if equal, false if not.
*/
public function a_compareType($data, $compareTo)
{
	$dataType = gettype($data);
	if ($dataType === gettype($compareTo))
	{
		switch($dataType)
		{
		case 'string':
		case 'integer':
		case 'double':
		case 'boolean':
		case 'NULL':
			if ($data === $compareTo)
			{
				return true;
			}

			break;

		case 'array':
			return a_identicalArrays($data, $compareTo);

		case 'object':
			if (get_class($data) === get_class($compareTo))
			{
				return true;
			}

			break;

		case 'resource':
			if (@get_resource_type($data) ===
			    @get_resource_type($compareTo))
			{
				return true;
			}

			break;

		default:
			break;
		}
	}

	return false;
}
```

```
/**
 * a_showData
 *
 * Output the value of the (optionally) named variable
 * @param mixed $value = value to output
 * @param string $name = (optional) name of the variable
 */
public function a_showData($value, $name=null)
{
	$buffer = 'VALUE';
	if ($name)
	{
		$buffer .= sprintf(' of %s', $name);
	}

	$type = gettype($value);
	$buffer .= sprintf(': (%s)', $type);

	switch($type)
	{
	case 'string':
		$buffer .= (string)$value;
		break;

	case 'boolean':
		$buffer .= sprintf('%b', $value);
		break;

	case 'integer':
		$buffer .= sprintf('%d', $value);
		break;

	case 'double':
	case 'float':
		$buffer .= sprintf('%f', $value);
		break;

	case 'array':
		$buffer .= \Library\PrintU\FormatArray::format($value);
		break;

	case 'object':
		$buffer .= get_class($value);
		break;

	case 'resource':
		$buffer .= get_resource_type($value);
		break;

	case 'NULL':
		$buffer .= 'null';
		break;

	case 'unknown type':
	default:
		$buffer .= sprintf('%s', $value);
		break;
	}

	$this->assertLogMessage($buffer);
}
```

```
/**
 * a_exceptionCaughtTrue
 *
 * check the exception caught for true status
 * @param boolean $exit = true to report and exit on failure, false to return
 */
public function a_exceptionCaughtTrue($exit=true)
{
	$this->labelBlock('Exception caught true.', 40, '*');

	$assertion = '$this->exceptionCaught()';
	if (! $this->assertTrue($assertion, sprintf('Asserting: %s', $assertion)))
	{
		$this->a_printException();
		if ($exit)
		{
			$this->a_outputAndDie();
		}
	}
}

/**
 * a_exceptionCaughtFalse
 *
 * check the exception caught for false status
 * @param boolean $exit = true to report and exit on failure, false to return
 */
public function a_exceptionCaughtFalse($exit=true)
{
	$this->labelBlock('Exception caught false.', 40, '*');

	$assertion = '$this->exceptionCaught()';
	if (! $this->assertFalse($assertion,
				sprintf('Falsely asserting: %s', $assertion)))
	{
		$this->a_printException();
		if ($exit)
		{
			$this->a_outputAndDie();
		}
	}
}

/**
 * a_printException
 *
 * Output the exception to the log as a printable string
 */
public function a_printException()
{
	$this->assertLogMessage((string)$this->exception);
}
```

```
	/**
	 * a_absoluteFileName
	 *
	 * Takes a workspace relative file name and returns an absolute path to the file
	 * @param string $name = workspace relative file name
	 * @return string $name = absolute path to the file
	 */
	public function a_absoluteFileName($name)
	{
		if (! stream_is_local($name))
		{
			return $this->a_fileName = $name;
		}

		$this->a_fileName = str_replace(array('/', '\\'),
							DIRECTORY_SEPARATOR, $name);

		if (substr($this->a_fileName, 0, 1) != DIRECTORY_SEPARATOR)
		{
			$this->a_fileName = sprintf("%s%s%s",
						$this->properties['RootPath'],
						DIRECTORY_SEPARATOR, $this->a_fileName);
		}

		return $this->a_fileName;
	}

	/**
	 * a_outputAndDie
	 *
	 * Output a message and terminate the program
	 * @param string $message = (optional) message to output
	 */
	public function a_outputAndDie($message='')
	{
		if ($message != '')
		{
			$this->assertLogMessage($message);
			$this->properties->Logger_Object->log($message, 'error');
		}

		throw new Exception($message, \Library\Error::code('PhpTestingFailure'));
	}

}
```