EcomDev_PHPUnit Manual

version 0.2.0

Table Of Contents

Introduction

Magento is a quite complex platform without built in unit test suite, so the code is not oriented on running tests over it. This extension was created especially for resolving this problem and promoting test driven development practices in Magento developers community.

It doesn't changes core files or brakes your Magento installment database, because all the system objects are replaced during the run-time with the test ones and a separate database connection is used for tests.

Minimal System Requirements

- **PHP 5.3 or greater**
- **PHPUnit 3.5**
- **Magento CE 1.4.x, CE 1.5.x, PE 1.9.x, PE 1.10.x, EE 1.9.x, EE 1.10.x**

Installation

1. Checkout Extension from our public SVN (a) or install it via MagentoConnect (b)
   a) http://svn.ecomdev.org/svn/ecomdev-phpunit/tags/0.2.0

- Copy extension directories into your Magento installment root

   b) http://www.magentocommerce.com/magento-connect/Ecommerce%20Developers/extension/5717/ecomdev_phpunit

2. Open **app/etc/local.xml.phpunit** in editor that you are comfortable with
   a) Specify database credentials that will be used for test suite in **global/resources/default_setup/connection** node
   b) Specify **base_url** for **secure** and **unsecure** requests in **default/web** node. It required for proper controller tests.
3. Open console and navigate to your Magento root directory.
4. Run the unit tests first time for installing test database. It will take about 3 minutes:
5. If it shows that there was no tests found, it means that extension was successfully installed. If it shows some errors than it means that your customizations has install scripts that relay on your current database data so you should fix them.

Setting Up Your Module

For letting system know that your module contains unit tests you just need to add few lines into your module **config.xml** file:

```
<phpunit>
        <suite>
            <modules>
                    <Your_ModuleName />
            </modules>
         </suite>
</phpunit>
```

Than you need to create **Test** directory in your module that will be used for searching of the test files. This directory can contain such sub-directories that are also test groups:

- **Model** for test cases related to models
- **Block** for test cases related to blocks that can be tested in isolation
- **Helper** for test cases related to helpers
- **Controller** for test cases related to controller actions together with layout and blocks
- **Config** for test cases related to your module configuration file definitions

Your test case class should be named in such a way:

[Your Module]_Test_[Group Directory]_[Related Entity Name]

For instance if you want to test your custom product model in module called "MyCustom_Module" you need to create a test case class with such name:

MyCustom_Module_Test_Model_Product

and extended from **EcomDev_PHPUnit_Test_Case** class

Group directories can have as many nested levels as you wish, system collects test case files recursively.

Writing Tests

When you already setup the test module it is time for starting writing your tests.

There is three test case classes from which you can extend yours:

- **EcomDev_PHPUnit_Test_Case** is for testing models, blocks and helpers
- **EcomDev_PHPUnit_Test_Case_Config** is for testing your module configuration (e.g. models, blocks, helpers definitions, events observer definitions, layout file existence, etc)
- **EcomDev_PHPUnit_Test_Case_Controller** is for testing your controller actions and layout rendering process.

Base Test Case

Lets look at this small example test case:

```
class EcomDev_Example_Test_Model_Product extends EcomDev_PHPUnit_Test_Case
{
    /**
     * Product price calculation test
     *
     * @test
     * @loadFixture
     * @doNotIndexAll
     * @dataProvider dataProvider
     */
    public function priceCalculation($productId, $storeId)
    {
        $storeId = Mage::app()->getStore($storeId)->getId();
        /* @var $product Mage_Catalog_Model_Product */
        $product = Mage::getModel('catalog/product')
            ->setStoreId($storeId)
            ->load($productId);
        $expected = $this->expected('%s-%s', $productId, $storeId);
        // Check that final price
        // is the minimal one for the product
        $this->assertEquals(
            $expected->getFinalPrice(),
            $product->getFinalPrice()
        );
        // Check that base price is proper value
```

```
            // for the current website
            $this->assertEquals(
                $expected->getPrice(),
                $product->getPrice()
            );
        }
    }
```

This code is a test case with a test called **priceCalculation,** it checks that product price calculation works correctly for a particular product. I suppose you have noticed that it uses **“@loadFixture”** and **“@dataProvider dataProvider”** annotations. They are used for loading of fixture and data provider YAML files, that contains required data for test. Fixture file will load data into database tables or change configuration nodes for this test. Default **dataProvider** method loads data for different test calls, that makes it more flexible.

Also test case uses **expected()** method for retrieving expectations for test case, they are also placed in YAML file. In combination with dataProvider it makes test case code more flexible for testing product prices and you do not have hard-coded values.

All of these YAML files should be placed in the directory that has the same name as php file with the test case and in the same directory path. For instance for this test case it will be in **EcomDev/Example/Test/Model/Product/** where “**EcomDev/Example/Test/Model”**is our test case directory and “**Product**” is directory with YAML files. Each file type has own subdirectory:

- **expectations** for expected data files
- **fixtures** for test fixture files
- **providers** for data provider files

Fixtures and expectations allow custom file naming if you pass its name to **@loadFixture** and **@loadExpectation** annotations. If file name is not specified by annotations is presented, then will be used the name of the test method. For example for our test it will be **priceCalculation.yaml**.

Test Case supports loading of more then one fixture and expectations files. For making this possible just create one more annotation with the file name. The files will be merged into one big structure.

Fixtures

Now let take a look at how fixture file works. Here is an example of file for test case we saw before:

EcomDev/Example/Test/Model/Product/fixtures/priceCalculation.yaml

```
scope:
  website: # Initialize websites
    - website_id: 2
      code: usa_website
      name: USA Website
      default_group_id: 2
    - website_id: 3
      code: canada_website
      name: Canada Website
      default_group_id: 3
    - website_id: 4
      code: german_website
      name: German Website
      default_group_id: 4
  group: # Initializes store groups
    - group_id: 2
      website_id: 2
      name: USA Store Group
      default_store_id: 2
      root_category_id: 2 # Default Category
    - group_id: 3
      website_id: 3
      name: Canada Store Group
      default_store_id: 3
      root_category_id: 2 # Default Category
    - group_id: 4
      website_id: 4
      name: German Store Group
      default_store_id: 4
      root_category_id: 2 # Default Category
  store: # Initializes store views
    - store_id: 2
      website_id: 2
      group_id: 2
      code: usa
      name: USA Store
      is_active: 1
    - store_id: 3
      website_id: 3
      group_id: 3
      code: canada
      name: Canada Store
      is_active: 1
    - store_id: 4
      website_id: 4
      group_id: 4
      code: germany
      name: Germany Store
      is_active: 1
```

```
config:
  default/catalog/price/scope: 1 # Set price scope to website
eav:
  catalog_product:
    - entity_id: 1
      type_id: simple
      sku: book
      name: Book
      short_description: Book
      description: Book
      url_key: book
      stock:
        qty: 100.00
        is_in_stock: 1
      website_ids:
        - usa_website
        - canada_website
        - german_website
      category_ids:
        - 2 # Default Category
      price: 12.99
      tax_class_id: 2 # Taxable Goods
      status: 1            # Enabled
      visibility: 4        # Visible in Catalog & Search
      /websites:           # Set different prices per website
        usa_website:
          special_price: 9.99
        german_website:
          price: 9.99
          special_price: 5.99
```

This file loads websites, store groups, store views and products data into database for our test. Also it sets configuration values for allowing different price values per website for our test.

scope/website, scope/group, scope/store contain definition of websites, store groups and store view accordingly. **config/default/catalog/price/scope** set a value to configuration node with the same path. **eav/catalog_product** contains definition of product attributes data, that will be transformed into eav database scheme.

> All the fixture operations are processed in the same order as they are presented in the file, so place them in a proper order. For instance do not place product definitions with scope usage before creation of scope data, because it will result into the error.

Short File Format Reference

- **tables** loads database tables with data that is presented in the file, each key in this array is a table alias name, that is defined in confirmation and values is an array of table rows.

- **scope** contains data for scope creation (websites, store groups, store views). It works in the similar way as **tables** operation, but additional initializes newly created scope entities. Contains only three array keys, it is **website**, **group** (for store groups) and **store** (for store views). Values inside of these array keys the same as in table row definitions.
  For setting current store view for a test case use **setCurrentStore()** method that accepts store code or store id as a parameter.

```
$this->setCurrentStore('default');
```

- **config** contains configuration path values that should be changed for current test. Array key is the path in configuration file. Array value is of a scalar type.
- **configXml** works in the same way as **config,** but allows to replace configuration path with an XML value.
- **eav** is one of the most interesting operations in the fixture file, it allows load data for EAV entity and if it requires some index process it runs it. Array key is an entity type code and values is an array of attribute values. Some entities support special keys for multi-store values functionality. For instance **catalog_product** and **catalog_category** support two keys **"/websites"** and **"/stores"** for specifying attribute data additionally for a website or store view. In "/websites" array key a website code and in "/stores" it is store code accordingly.
  Also you can extend EAV Fixture loader by adding custom loaders for entity types, you just need create a resource model and extend it from **EcomDev_PHPUnit_Model_Mysql4_Fixture_Eav_Abstract**, implement your custom process of load and add your resource model class alias into configuration:

```
<config>
   <phpunit>
        <suite>
            <fixture>
                <[entity_type_code]>[your_module/model]</[entity_type_code]>
            </fixture>
        </suite>
   </phpunit>
</config>
```

  Since some of the test doesn't require indexes to be rebuild for EAV entities, there is an ability for switching them off in annotations, it improves the speed of test execution:

  - **@doNotIndex [index_code]** for switching a particular index

- **@doNotIndexAll** for switching off all indexes for EAV entities.

Expectations

Expectations file in opposite to fixtures does not have any special operations or fixed file structure. When you call **expected()** method it returns data from file in the requested path.

For instance our expectation data for the test case is:

EcomDev/Example/Test/Model/Product/expectations/priceCalculation.yaml

```
1-2:  # Product=Book Store=USA
  final_price: 9.99
  price: 12.99
1-3:  # Product=Book Store=Canada
  final_price: 12.99
  price: 12.99
1-4:  # Product=Book Store=Germany
  final_price: 5.99
  price: 9.99
```

Then this calls will returns will returns the following:

```
$this->expected('1-2')->getFinalPrice(); // 9.99
$this->expected('%s-%s', 1, 3)->getFinalPrice(); // 12.99
$this->expected()->getData(); // array('1-2' => array('final_price' => 9.99) /* etc.. */)
```

Data Providers

Here is an expectation file for our test case:

EcomDev/Example/Test/Model/Product/providers/priceCalculation.yaml

```
-
  - 1
  - usa
-
  - 1
  - canada
-
  - 1
  - germany
```

Data provider file has also very simple structure. Each item on the first level is a particular method call with arguments. By the data provided in this file, PHPUnit will create such test calls:

```
EcomDev_Example_Test_Model_Product::priceCalculation(1, 'usa');
EcomDev_Example_Test_Model_Product::priceCalculation(1, 'canada');
EcomDev_Example_Test_Model_Product::priceCalculation(1, 'germany');
```

So as you see using of expectation in combination with data providers allows you to minimize amount of code in the test and provide flexibility for different cases just by specifying different input data.

Test Doubles

So what happens if your module uses classes from another one and you just want to check that particular methods was used by your code, then you should use test doubles. If you are not familiar with them, please read this chapter from PHPUnit manual:

http://www.phpunit.de/manual/current/en/test-doubles.html

The extension provides you a set of methods for easier test doubles creation:

- **getModelMock()** creates a new mock object by class alias. Its arguments
  - **string $classAlias** is a class alias of the model (e.g. 'catalog/product')
  - **array $methods** list of methods that should be mocked, if is an empty array then all the methods will be mocked. [optional]
  - **boolean $isAbstract** a flag the indicates if specified model is an abstract class, so all the abstract methods will be mocked. [optional]
  - **array $constructorArguments** is a list of arguments that will be passed to model constructor. [optional]
  - **string $mockClassAlias** the class alias that will be used as the mock one. [optional]
  - **boolean $callOriginalConstructor** if this flag is set to **false**, it **__construct()** method will not be called in the original class. [optional]
  - **boolean $callOriginalClone** if this flag is set to **false**, it **__clone()** method will not be called in the original class. [optional]
  - **boolean $callAutoload** indicates if autoload should be used for checking class existence during creation of the mock object. **true** by default [optional]
- **getResourceModelMock()** is similar to **getModelMock(),** but returns a mock object

for a resource model

- **getBlockMock()** is similar to **getModelMock(),** but returns a mock object for a block
- **getHelperMock()** is similar to **getModelMock(),** but returns a mock object for a helper
- **getModelMockBuilder()** returns a mock builder object for a model. Its has only one argument:
  - **string $classAlias** is a class alias of the model (e.g. 'catalog/product')
- **getResourceModelMockBuilder()** returns a mock builder object for a resource model. Its has only one argument:
  - **string $classAlias** is a class alias of the resource model (e.g. 'catalog/product_collection')
- **getBlockMockBuilder()** returns a mock builder object for a block. Its has only one argument:
  - **string $classAlias** is a class alias of the block (e.g. 'catalog/product_list')
- **getHelperMockBuilder()** returns a mock builder object for a helper. Its has only one argument:
  - **string $classAlias** is a class alias of the helper (e.g. 'catalog')
- **replaceByMock()** method replaces a Magento system resource with mock object. Here is the list of it arguments:
  - **string $type** is a type of system resource. Available values are:
    - **model** replaces call to **Mage::getModel()** and **Mage::getSingleton()**
    - **resource_model** replaces call to **Mage::getResourceModel()** and **Mage::getResourceSingleton()**
    - **singleton** replaces call to **Mage::getSingleton()**
    - **resource_singleton** replaces call to **Mage::getResourceSingleton()**
    - **helper** replaces call to **Mage::helper()**
    - **block** replaces call to **Mage::app()->getLayout()->createBlock()**
  - **string $classAlias** the class alias of the Magento resource
  - **PHPUnit_Framework_MockObject_MockObject | PHPUnit_Framework_MockObject_MockBuilder $mock** the mock object that will replace system resource.

For instance in controller test case there is created a test double for **core/cookie** singleton:

```
protected function registerCookieStub()
```

```
{
   $cookie = $this->getModelMock('core/cookie', array('set', 'delete'));

   $cookie->expects($this->any())
      ->method('set')
      ->will($this->returnCallback(
         array($this, 'setCookieCallback')
      ));

   $cookie->expects($this->any())
      ->method('delete')
      ->will($this->returnCallback(
         array($this, 'deleteCookieCallback')
      ));

   $this->replaceByMock('singleton', 'core/cookie', $cookie);
   return $this;
}
```

It is used there for managing cookies between different controller dispatch actions.

Assertions

Base test case contains some specific assertions that will be useful for Model, Block, Helper test:

- **assertEventDispatched()** asserts that a particular event was dispatched in the system. It has only one argument:
  - **string $eventName** is the name of the event that will be checked.
- **assertEventNotDispatched()** asserts that a particular event was not dispatched in the system. It has only one argument:
  - **string $eventName** is the name of the event that will be checked.
- **assertEventDispatchedExactly()** asserts that a particular event was dispatched in the system exact number of times. Its arguments:
  - **string $eventName** is the name of the event that will be checked.
  - **int $times** expected number of times when event was dispatched.
- **assertEventDispatchedAtLeast()** asserts that a particular event was dispatched in the system at least expected number of times. Its arguments:
  - **string $eventName** is the name of the event that will be checked.
  - **int $times** expected number of times when event was dispatched.
- **assertJson()** asserts that string is a valid JSON. Its arguments:

  - **string $string** string that will be checked
  - **string $message** custom assertion message [optional]
- **assertNotJson()** asserts that string is not a valid JSON. Its arguments:
  - **string $string** string that will be checked
  - **string $message** custom assertion message [optional]
- **assertJsonMatch()** asserts that string is valid JSON and matches its array representation. Its arguments:
  - **string $string** string that will be checked
  - **array $expectedValue** the array the will be used for matching JSON structure
  - **string $message** custom assertion message [optional]
  - **string $matchType** type JSON structure match. Supported values are:
    - **EcomDev_PHPUnit_Constraint_Json::MATCH_AND** matches that all array elements is presented in the JSON
    - **EcomDev_PHPUnit_Constraint_Json::MATCH_OR** matches that at least one array element is presented in the JSON
    - **EcomDev_PHPUnit_Constraint_Json::MATCH_EXACT** matches that array is the same as the JSON
- **assertJsonNotMatch()** asserts that string is not valid JSON and matches its array representation. Its arguments:
  - **string $string** string that will be checked
  - **array $expectedValue** the array the will be used for matching JSON structure
  - **string $message** custom assertion message [optional]
  - **string $matchType** type JSON structure match. Supported values are:
    - **EcomDev_PHPUnit_Constraint_Json::MATCH_AND** matches that all array elements is presented in the JSON
    - **EcomDev_PHPUnit_Constraint_Json::MATCH_OR** matches that at least one array element is presented in the JSON
    - **EcomDev_PHPUnit_Constraint_Json::MATCH_EXACT** matches that array is the same as the JSON

Config Test Case

As practice shows it is very important to test your module configuration, because at least 20% of issues are related to mistypes or not proper model/block/helper rewrite.

Here the list of assertions that are available for configuration test:

Class Alias Assertions

- **assertBlockAlias()** asserts that block alias is mapped to expected class name
  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertBlockAliasNot()** asserts that block alias is not mapped to expected class name
  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should not be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertModelAlias()** asserts that model alias is mapped to expected class name
  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertModelAliasNot()** asserts that model alias is not mapped to expected class name
  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should not be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertResourceModelAlias()** asserts that resource model alias is mapped to expected class name
  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertResourceModelAliasNot()** asserts that resource model alias is not mapped to expected class name

  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should not be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertHelperAlias()** asserts that helper alias is mapped to expected class name
  - **string $classAlias** is class alias that will be asserted, if helper name is not defined, like in **Mage::helper('catalog')** than it will automatically assert it as module_prefix/data (e.g. **'catalog/data'**).
  - **string $expectedClassName** is expected class name that should be returned after reading the configuration
  - **string $message** is custom assertions message [optional]
- **assertHelperAliasNot()** asserts that helper alias is not mapped to expected class name
  - **string $classAlias** is class alias that will be asserted
  - **string $expectedClassName** is expected class name that should not be returned after reading the configuration
  - **string $message** is custom assertions message [optional]

Some examples of usage:

```
$this->assertModelAlias('catalog/product', 'Mage_Catalog_Model_Product');
$this->assertModelAliasNot('catalog/product', 'Mage_Catalog_Model_Product_Type');
$this->assertResourceModelAlias('sales/order', 'Mage_Sales_Model_Mysql4_Order');
```

Module Assertions

- **assertModuleCodePool()** asserts that the module is in a particular code pool
  - **string $expected** expected code pool
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleDepends()** asserts that the module depends on another module
  - **string $requiredModuleName** module that should be mentioned in dependencies

- string $message custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleNotDepends()** asserts that the module does not depend on another one.
  - **string $requiredModuleName** module that should be mentioned in dependencies
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleVersion()** asserts that the module version is equal to expected one
  - **string $expectedVersion** expected version
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleVersionGreaterThan()** asserts that the module version is greater than expected one
  - **string $expectedVersion** expected version
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleVersionGreaterThanOrEquals()** asserts that the module version is greater than or equal to expected one
  - **string $expectedVersion** expected version
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleVersionLessThan()** asserts that the module version is less than expected one
  - **string $expectedVersion** expected version
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]
- **assertModuleVersionLessThanOrEquals()** Assert that the module version is less

than or equal to expected one

  - **string $expectedVersion** expected version
  - **string $message** custom assertion message [optional]
  - **string $moduleName** the module name that will be used for assertion. If not specified, then current module will be used [optional]

Some examples:

```
$this->assertModuleCodePool('local');
$this->assertModuleVersionLessThan('1.0');
$this->assertModuleVersionLessThanOrEquals('1.0');
$this->assertModuleVersion('0.1.0');
```

Config Node Assertions

- **assertConfigNodeHasChild()** asserts that configuration node has child with expected tag name
  - **string $nodePath** the configuration path
  - **string $childName** the expected child node name
  - **string $message** custom assertion message [optional]
- **assertConfigNodeNotHasChild()** asserts that configuration node does not have child with expected tag name
  - **string $nodePath** the configuration path
  - **string $childName** the expected child node name
  - **string $message** custom assertion message [optional]
- **assertConfigNodeHasChildren()** asserts that configuration node has children
  - **string $nodePath** the configuration path
  - **string $message** custom assertion message [optional]
- **assertConfigNodeNotHasChildren()** asserts that configuration node does not have children
  - **string $nodePath** the configuration path
  - **string $message** custom assertion message [optional]
- **assertConfigNodeValue()** asserts that configuration node value is equal to expected value

- ○ **string $nodePath** the configuration path
  - ○ **string $expectedValue** the expected value
  - ○ **string $message** custom assertion message [optional]
- **assertConfigNodeNotValue()** asserts that configuration node value is not equal to expected value
  - ○ **string $nodePath** the configuration path
  - ○ **string $expectedValue** the expected value
  - ○ **string $message** custom assertion message [optional]
- **assertConfigNodeLessThan()** asserts that configuration node value is less than expected number
  - ○ **string $nodePath** the configuration path
  - ○ **decimal $expectedValue** the expected value
  - ○ **string $message** custom assertion message [optional]
- **assertConfigNodeLessThanOrEquals()** asserts that configuration node value is less than or equals to expected number
  - ○ **string $nodePath** the configuration path
  - ○ **decimal $expectedValue** the expected value
  - ○ **string $message** custom assertion message [optional]
- **assertConfigNodeGreaterThan()** asserts that configuration node value is greater than expected number
  - ○ **string $nodePath** the configuration path
  - ○ **decimal $expectedValue** the expected value
  - ○ **string $message** custom assertion message [optional]
- **assertConfigNodeGreaterThanOrEquals()** asserts that configuration node value is greater than or equals to expected number
  - ○ **string $nodePath** the configuration path
  - ○ **decimal $expectedValue** the expected value
  - ○ **string $message** custom assertion message [optional]
- **assertConfigNodeContainsValue()** asserts that configuration node contains expected value is in comma separated value list
  - ○ **string $nodePath** the configuration path

- **scalar $expectedValue** the expected value
- **string $message** custom assertion message [optional]

- **assertConfigNodeNotContainsValue()** asserts that configuration node contains expected value is in comma separated value list
  - **string $nodePath** the configuration path
  - **scalar $expectedValue** the expected value
  - **string $message** custom assertion message [optional]
- **assertConfigNodeSimpleXml()** asserts that configuration node is equal to simple xml element value
  - **string $nodePath** the configuration path
  - **scalar $expectedValue** the expected value
  - **string $message** custom assertion message [optional]
- **assertConfigNodeNotSimpleXml()** asserts that configuration node is not equal to simple xml element value
  - **string $nodePath** the configuration path
  - **scalar $expectedValue** the expected value
  - **string $message** custom assertion message [optional]

Layout Assertions

- **assertLayoutFileDefined()** asserts that configuration has definition of the layout file
  - **string $area** the area of layout file. Possible values are **frontend** and **adminhtml**
  - **string $expectedFileName** expected layout file name, for instance **catalog.xml**
  - **string $layoutUpdate** if layout update name is specified, then it will restrict assertion by it. [optional]
  - **string $message** custom assertion message [optional]
- **assertLayoutFileExists()** asserts that layout file exists in current design package
  - **string $area** the area of layout file. Possible values are **frontend** and **adminhtml**
  - **string $expectedFileName** expected layout file name, for instance **catalog.xml**
  - **string $message** custom assertion message [optional]
- **assertLayoutFileExistsInTheme()** asserts that layout file exists in a particular theme and design package

- ◦ **string $area** the area of layout file. Possible values are **frontend** and **adminhtml**
  - ◦ **string $expectedFileName** expected layout file name, for instance **catalog.xml**
  - ◦ **string $theme** theme for searching layout file
  - ◦ **string $designPackage** design package for searching the layout file [optional]
  - ◦ **string $message** custom assertion message [optional]

Some examples:

```
$this->assertLayoutFileDefined('frontend', 'catalog.xml', 'catalog');
$this->assertLayoutFileExists('frontend', 'catalog.xml');
```

Event Observer Assertions

- **assertEventObserverDefined()** asserts that event observer is defined for an event and not disabled. If observer name is defined, it additionaly checks it.
  - **string $area** the area of event observer definition, possible values are **global**, **frontend**, **adminhtml**
  - **string $eventName** is the name of the event that should be observed
  - **string $observerClassAlias** observer class alias, for instance **catalog/observer**
  - **string $observerMethod** the method name that should be invoked for
  - **string $observerName** [optional]
  - **string $message** custom assertion message [optional]
- **assertEventObserverNotDefined()** asserts that event observer is not defined for an event or disabled.
  - **string $area** the area of event observer definition, possible values are **global**, **frontend**, **adminhtml**
  - **string $eventName** is the name of the event that should be observed
  - **string $observerClassAlias** observer class alias, for instance **catalog/observer**
  - **string $observerMethod** the method name that should be invoked for
  - **string $observerName** [optional]
  - **string $message** custom assertion message [optional]

Some examples:

```
$this->assertEventObserverDefined(
    'frontend', 'customer_login', 'catalog/product_compare_item', 'bindCustomerLogin'
);

$this->assertEventObserverDefined(
    'frontend', 'customer_login',
    'catalog/product_compare_item', 'bindCustomerLogin', 'catalog'
);
```

Controller Test Case

For testing controllers there are created a special test case that allows you to dispatch a particular actions and evaluate the actions that was performed.

Here are some examples of different test controller cases:

Homepage test

```
$this->dispatch('');
$this->assertLayoutHandleLoaded('cms_index_index');
$this->assertLayoutBlockCreated('left');
$this->assertLayoutBlockCreated('right');
$this->assertLayoutBlockRendered('content');
$this->assertLayoutBlockTypeOf('left', 'core/text_list');
$this->assertLayoutBlockNotTypeOf('left', 'core/links');
```

Customer account login test

```
$this->dispatch('customer/account/login');
$this->assertRequestRoute('customer/account/login');
$this->assertLayoutHandleLoaded('customer_account_login');
$this->assertLayoutHandleNotLoaded('cms_index_index');
$this->assertResponseBodyContains('login or create');
```

Testing footer links adding actions:

```
•   $this->dispatch('');
    $this->assertLayoutBlockActionNotInvoked('footer_links', 'addLink', '', array('Custom Title'));
    $this->assertLayoutBlockActionInvokedAtLeast('footer_links', 'addLink', 4, '');
```

```
$this->assertLayoutBlockActionInvokedAtLeast('footer_links', 'addLink', 1, '', array('Site Map'));
// Assertions of sitemap link is added
$this->assertLayoutBlockActionInvokedAtLeast('footer_links', 'addLink', 1, '', array(
        'label' => 'Site Map', 'url' => Mage::helper('catalog/map')->getCategoryUrl()));
```

Setting custom parameters to request

```
$this->getRequest()->setQuery('id' => $productId));
$this->getRequest()->setHeader('User-Agent', $userAgent);
// Dispatches product view action in default store
$this->dispatch('catalog/product/view', array('_store' => 'default');
```

In the controller test case used test **EcomDev_PHPUnit_Controller_Request_Test** and **EcomDev_PHPUnit_Controller_Response_Test** objects that was extended from **Mage_Core_Controller_Request_Http** and **Mage_Core_Controller_Response_Http** accordingly. There are available some special methods in all of them:

Test Case Methods

- **getRequest()** returns test request object, that can be changed before calling **dispatch()**
- **getResponse()** returns test response object, that will be analyzed by test case
- **getLayout()** returns test layout model, that will be analyzed by test case
- **getCookies()** returns cookie storage object that is used for emulation of the browser behavior. This object is an instance of **Zend_Http_CookieJar**.
- **reset()** resets request, response, layout objects and $_SESSION superglobal

Request Methods

- **reset()** resets request object to its defaults
- **setCookie()** sets a cookie value for current request
  - **string $name** cookie name
  - **string $value** cookie value
- **setCookies()** sets a cookie value for current request
  - **array $cookies** array of cookie key value pair
- **resetCookies()** resets all the cookies that was set for request

- **resetPost()** resets all the post data that was set for request
- **resetParams()** resets user prams that was set for request
- **resetInternalProperties()** resets internal properties to its default values
- **setHeader()** sets a header value for current request
  - **string $name** header name
  - **string $value** header value
- **setHeaders()** set a list of the headers for the request
  - **array $headers** array of header key value pair
- **resetHeaders()** resets the headers that was set for a test request
- **setServer()** sets value for a particular $_SERVER superglobal array key for test request
  - **string $name** server variable name
  - **string $value** server variable value
- **setServers()** sets multiple values for $_SERVER superglobal in test request
  - **array $servers** array of server variable key value pair
- **resetServer()** resets $_SERVER superglobal to a previous state, before its first change.
- **setMethod()** sets request method for test request
  - **string $method** desired request method. Available values are: **POST, GET, PUT, HEAD, OPTIONS, DELETE**
- **setIsSecure()** sets server variable **HTTPS** to **on** or to **null**
  - **boolean $flag** indicates whether **HTTPS** is **on** or not

Response Methods

- **reset()** resets response object to its initial state
- **getSentHeaders()** returns rendered headers array that were sent by **sendHeaders()** method, if headers was not sent, then returns **null**
- **getSentHeader()** returns a particular header that was sent or **false** if it wasn't.
  - **string $headerName** the header name that should be sent
- **getOutputBody()** returns the content that was rendered by **outputBody()** method during **sendRequest()** call.

- **getSentResponse()** returns rendered response by **sendResponse()** method, if response was not sent, then it returns **null**

Layout Methods

- **reset()** resets layout instance to its defaults
- **getRecords()** returns an array of records with layout operations that were executed. The structure of the resulting array like the following

  ```
  array('action' => array('target' => array(array($param1, $param2 /*, etc */)));
  ```

  Where **action** is an action type that was performed on the **target**. In the **target** array list of all actions with parameters for it. Possible actions are:

  - **EcomDev_PHPUnit_Model_Layout::ACTION_HANDLE_LOADED** for records about loading of a handle into layout updates.
  - **EcomDev_PHPUnit_Model_Layout::ACTION_BLOCK_CREATED** for records about creation of a particular block in layout
  - **EcomDev_PHPUnit_Model_Layout::ACTION_BLOCK_RENDERED** for records about rendering of a particular block
  - **EcomDev_PHPUnit_Model_Layout::ACTION_BLOCK_REMOVED** for records about removing of a particular block from layout
  - **EcomDev_PHPUnit_Model_Layout::ACTION_BLOCK_ACTION** for records about performed block method calls with **<action method="" />** operation. In this case parameters are processed method arguments.
  - **EcomDev_PHPUnit_Model_Layout::ACTION_RENDED** used for recording that layout rendering method **getOutput()** was called.
- **findAll()** returns an array of all performed actions on a target, or if target not specified it returns all actions of a particular type
  - **string $action** an action type that was recorded
  - **string|null $target** action target [optional]
- **findAllTargets()** returns a list of targets that were used in action with specified type
  - **string $action** an action type that was recorded
- **findFirst()** returns first action that was recorded for action type on a particular target
  - **string $action** an action type that was recorded
  - **string $target** action target

- **findByParameters()** returns actions that were recorded for action type on a particular target and matches parameters condition
    - **string $action** an action type that was recorded
    - **string $target** action target
    - **array $parameters** parameters for matching the record
    - **string $searchType** type of parameters search matching, defines logic for calculation of parameters intersection with the record. Possible types:
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_OR** at least one parameter exists in a record
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_AND** all parameters exist in a record
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_EXACT** parameters are the same in record
- **isLoaded()** checks that layout was loaded

Assertions

List of available assertions for controller test case

Request Routing

For testing request routing process, you can use such methods:

- **assertRequestDispatched()** asserts that controller request is dispatched
    - **string $message** custom assertion message [optional]
- **assertRequestNotDispatched()** asserts that controller request is dispatched
    - **string $message** custom assertion message [optional]
- **assertRequestForwarded()** asserts that controller request is forwarded
    - **string $message** custom assertion message [optional]
- **assertRequestNotForwarded()** asserts that controller request is not forwarded
    - **string $message** custom assertion message [optional]
- **assertRequestRoute()** asserts that current request route is matched expected one
    - **string $expectedRoute** expected route (**route_name/controller/action**), any route part can be a wildcard
    - **string $message** custom assertion message [optional]

- **assertRequestRouteNot()** asserts that current request route is not matched expected one
  - **string $expectedRoute** expected route (**route_name/controller/action**), any route part can be a wildcard
  - **string $message** custom assertion message [optional]
- **assertRequestRouteName()** asserts that current request route name is the same as expected
  - **string $expectedRouteName** expected route name, i.e. the name of the node in controllers config definition
  - **string $message** custom assertion message [optional]
- **assertRequestRouteNameNot()** asserts that current request route name is not the same as expected
  - **string $expectedRouteName** expected route name, i.e. the name of the node in controllers config definition
  - **string $message** custom assertion message [optional]
- **assertRequestControllerName()** asserts that current request controller name is the same as expected
  - **string $expectedControllerName** expected controller name
  - **string $message** custom assertion message [optional]
- **assertRequestControllerNameNot()** asserts that current request controller name is not the same as expected
  - **string $expectedControllerName** expected controller name
  - **string $message** custom assertion message [optional]
- **assertRequestControllerModule()** asserts that current request controller module is the same as expected
  - **string $expectedControllerModule** expected controller module
  - **string $message** custom assertion message [optional]
- **assertRequestControllerModuleNot()** asserts that current request controller module is not the same as expected
  - **string $expectedControllerModule** expected controller module
  - **string $message** custom assertion message [optional]
- **assertRequestActionName()** asserts that current request action name is the same as expected

  - **string $expectedActionName** expected action name
  - **string $message** custom assertion message [optional]
- **assertRequestActionNameNot()** asserts that current request action name is not the same as expected
  - **string $expectedActionName** expected action name
  - **string $message** custom assertion message [optional]
- **assertRequestBeforeForwardedRoute()** asserts that current request before forwarded route is matched expected
  - **string $expectedBeforeForwardedRoute** expected route (**route_name/controller/action**), any route part can be a wildcard
  - **string $message** custom assertion message [optional]
- **assertRequestBeforeForwardedRouteNot()** asserts that current request before forwarded route is not matched expected
  - **string $expectedBeforeForwardedRoute** expected route (**route_name/controller/action**), any route part can be a wildcard
  - **string $message** custom assertion message [optional]

Response General

For being sure that you controller action sets proper HTTP response code or redirects user to another url or internal Magento page, use these methods:

- **assertResponseHttpCode()** asserts HTTP response code value
  - **string $code** expected HTTP response code
  - **string $message** custom assertion message [optional]
- **assertRedirect()** asserts that response HTTP code is between 300 and 307
  - **string $message** custom assertion message [optional]
- **assertNotRedirect()** asserts that response HTTP code is not between 300 and 307
  - **string $message** custom assertion message [optional]
- **assertRedirectTo()** asserts that **Location** header value equals to url that is the same as specified Magento route. Internally calls **assertRedirectToUrl()** method
  - **string $route** expected route **(route_name/controller/action)**, any route part can be a wildcard
  - **array $params** route parameters

- **string $message** custom assertion message [optional]

- **assertRedirectToUrl()** asserts that **Location** header value equals to expected url. Internally calls **assertRedirect()** method.
  - **string $url** expected redirect url
  - **string $message** custom assertion message [optional]
- **assertRedirectToUrlStartsWith()** asserts that **Location** header value starts with expected url part. Internally calls **assertRedirect()** method.
  - **string $urlPart** expected url part
  - **string $message** custom assertion message [optional]
- **assertRedirectToUrlContains()** asserts that **Location** header value contains expected url part. Internally calls **assertRedirect()** method.
  - **string $urlPart** expected url part
  - **string $message** custom assertion message [optional]
- **assertRedirectToUrlRegExp()** asserts that **Location** header value matches PCRE pattern. Internally calls **assertRedirect()** method.
  - **string $pcrePattern** PCRE pattern for matching redirect url
  - **string $message** custom assertion message [optional]

Response Headers

Sometimes you need to check what headers were sent by your controller. These methods will help you:

- **assertResponseHeaderSent()** asserts that expected header was sent during the dispatch
  - **string $headerName** header name for assertion
  - **string $message** custom assertion message [optional]
- **assertResponseHeaderNotSent()** asserts that expected header was not sent during the dispatch
  - **string $headerName** header name for assertion
  - **string $message** custom assertion message [optional]
- **assertResponseHeader()** asserts that header value is evaluated by specified constraint
  - **string $headerName** header name for assertion

- **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for header value assertion
- **string $message** custom assertion message [optional]

- **assertResponseHeaderNot()** asserts that header value is not evaluated by specified constraint
  - **string $headerName** header name for assertion
  - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for header value assertion
  - **string $message** custom assertion message [optional]
- **assertResponseHeaderEquals()** asserts that header value is equal to expected one
  - **string $headerName** header name for assertion
  - **mixed $expectedValue** expected header value
  - **string $message** custom assertion message [optional]
  - **float $delta** delta for comparing float values [optional]
  - **integer $maxDepth** the maximum depth for comparing nested level structures (multidimensional arrays, objects, etc) [optional]
  - **boolean $canonicalize** indicates if it is required to sort compared arrays or replace strings line endings for comparing similar values [optional]
  - **boolean $ignoreCase** indicates if strings should be compared in case insensitive mode [optional]
- **assertResponseHeaderNotEquals()** asserts that header value is not equal to expected one
  - **string $headerName** header name for assertion
  - **mixed $expectedValue** expected header value
  - **string $message** custom assertion message [optional]
  - **float $delta** delta for comparing float values [optional]
  - **integer $maxDepth** the maximum depth for comparing nested level structures (multidimensional arrays, objects, etc) [optional]
  - **boolean $canonicalize** indicates if it is required to sort compared arrays or replace strings line endings for comparing similar values [optional]
  - **boolean $ignoreCase** indicates if strings should be compared in case insensitive mode [optional]
- **assertResponseHeaderSame()** asserts that response header value is the same as

expected one

  - **string $headerName** header name for assertion
  - **mixed $expectedValue** expected header value
  - **string $message** custom assertion message [optional]
- **assertResponseHeaderNotSame()** asserts that response header value is not the same as expected one
  - **string $headerName** header name for assertion
  - **mixed $expectedValue** expected header value
  - **string $message** custom assertion message [optional]
- **assertResponseHeaderContains()** asserts that response header value contains expected string
  - **string $headerName** header name for assertion
  - **string $expectedValue** expected string
  - **string $message** custom assertion message [optional]
  - **boolean $ignoreCase** indicates if strings should be matched in case insensitive mode [optional]
- **assertResponseHeaderNotContains()** asserts that response header value does not contain expected string
  - **string $headerName** header name for assertion
  - **string $expectedValue** expected string
  - **string $message** custom assertion message [optional]
  - **boolean $ignoreCase** indicates if strings should be matches in case insensitive mode [optional]
- **assertResponseHeaderRegExp()** asserts that response header matches specified PCRE pattern
  - **string $headerName** header name for assertion
  - **string $pcrePattern** PCRE pattern for assertion
  - **string $message** custom assertion message [optional]
- **assertResponseHeaderNotRegExp()** asserts that response header does not match specified PCRE pattern
  - **string $headerName** header name for assertion
  - **string $pcrePattern** PCRE pattern for assertion

    - **string $message** custom assertion message [optional]

Response Body

If your controller does not use layout rendering system or you want to check response body itself, then use these methods:

- **assertResponseBody()** asserts that response body value is evaluated by specified constraint
    - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for body assertion
    - **string $message** custom assertion message [optional]
- **assertResponseBodyNot()** asserts that response body value is not evaluated by specified constraint
    - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for body assertion
    - **string $message** custom assertion message [optional]
- **assertResponseBodyContains()** asserts that response body contains expected string
    - **string $expectedValue** expected string
    - **string $message** custom assertion message [optional]
    - **boolean $ignoreCase** indicates if strings should be matched in case insensitive mode [optional]
- **assertResponseBodyNotContains()** asserts that response body does not contain expected string
    - **string $expectedValue** expected string
    - **string $message** custom assertion message [optional]
    - **boolean $ignoreCase** indicates if strings should be matched in case insensitive mode [optional]
- **assertResponseBodyRegExp()** asserts that response body matches specified PCRE pattern
    - **string $pcrePattern** PCRE pattern for assertion
    - **string $message** custom assertion message [optional]
- **assertResponseBodyNotRegExp()** asserts that response body does not match specified PCRE pattern
    - **string $pcrePattern** PCRE pattern for assertion

    - **string $message** custom assertion message [optional]
- **assertResponseBodyJson()** asserts that response body is valid JSON.
    - **string $message** custom assertion message [optional]
- **assertResponseBodyNotJson()** asserts that response body is not valid JSON.
    - **string $message** custom assertion message [optional]
- **assertResponseBodyJsonMatch()** asserts that response body is valid JSON and matches its array representation.
    - **array $expectedValue** the array the will be used for matching JSON structure
    - **string $message** custom assertion message [optional]
    - **string $matchType** type JSON structure match. Supported values are:
        - **EcomDev_PHPUnit_Constraint_Json::MATCH_AND** matches that all array elements is presented in the JSON
        - **EcomDev_PHPUnit_Constraint_Json::MATCH_OR** matches that at least one array element is presented in the JSON
        - **EcomDev_PHPUnit_Constraint_Json::MATCH_EXACT** matches that array is the same as the JSON
- **assertResponseBodyJsonNotMatch()** asserts that response body is not valid JSON and matches its array representation.
    - **array $expectedValue** the array the will be used for matching JSON structure
    - **string $message** custom assertion message [optional]
    - **string $matchType** type JSON structure match. Supported values are:
        - **EcomDev_PHPUnit_Constraint_Json::MATCH_AND** matches that all array elements is presented in the JSON
        - **EcomDev_PHPUnit_Constraint_Json::MATCH_OR** matches that at least one array element is presented in the JSON
        - **EcomDev_PHPUnit_Constraint_Json::MATCH_EXACT** matches that array is the same as the JSON

Layout General

These methods created to testing layout loading processes:

- **assertLayoutLoaded()** asserts that layout was loaded
    - **string $message** custom assertion message [optional]

- **assertLayoutNotLoaded()** asserts that layout was not loaded
  - **string $message** custom assertion message [optional]
- **assertLayoutRendered()** asserts that layout was rendered
  - **string $message** custom assertion message [optional]
- **assertLayoutNotRendered()** asserts that layout was not rendered
  - **string $message** custom assertion message [optional]
- **assertLayoutHandleLoaded()** asserts that layout handle was loaded into layout updates
  - **string $handle** layout handle name
  - **string $message** custom assertion message [optional]
- **assertLayoutHandleNotLoaded()** asserts that layout handle was not loaded into layout updates
  - **string $handle** layout handle name
  - **string $message** custom assertion message [optional]
- **assertLayoutHandleLoadedAfter()** asserts that layout handle was loaded after another one
  - **string $handle** layout handle name
  - **string $after** another handle name
  - **string $message** custom assertion message [optional]
- **assertLayoutHandleLoadedBefore()** asserts that layout handle was loaded before another one
  - **string $handle** layout handle name
  - **string $before** another handle name
  - **string $message** custom assertion message [optional]

Layout Block General

This set of methods allows testing creation, removal or rendering of a particular block.

- **assertLayoutBlockCreated()** asserts that layout block is created and not removed with another layout instruction
  - **string $blockName** block name in layout
  - **string $message** custom assertion message [optional]

- **assertLayoutBlockRemoved()** asserts that layout block is removed via **<remove />** layout node
    - **string $blockName** block name in layout
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockRendered()** asserts that layout block is rendered via **toHtml()** method call
    - **string $blockName** block name in layout
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockNotRendered()** asserts that layout block is not rendered via **toHtml()** method call
    - **string $blockName** block name in layout
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockRenderedContent()** asserts that block rendered content is evaluated by specified constraint
    - **string $blockName** block name in layout
    - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for block content assertion
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockRenderedContentNot()** asserts that block rendered content is not evaluated by specified constraint
    - **string $blockName** block name in layout
    - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for block content assertion
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockTypeOf()** asserts that layout block type is an expected class alias
    - **string $blockName** block name in layout
    - **string $classAlias** block class alias that should be specified in type attribute
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockNotTypeOf()** asserts that layout block type is not an expected class alias
    - **string $blockName** block name in layout
    - **string $classAlias** block class alias that should not be specified in type attribute

- **string $message** custom assertion message [optional]

- **assertLayoutBlockInstanceOf()** asserts that layout block is an instance of expected class name
    - **string $blockName** block name in layout
    - **string $className** class name for assertion
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockNotInstanceOf()** asserts that layout block is not an instance of expected class name
    - **string $blockName** block name in layout
    - **string $className** class name for assertion
    - **string $message** custom assertion message [optional]

Layout Block Position

If you need to be sure that your block is in proper parent one or its positions in the sibling, use these methods:

- **assertLayoutBlockParentEquals()** asserts that layout block parent is equal to expected one
    - **string $blockName** block name in layout
    - **string $parentBlockName** parent block name in layout
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockParentNotEquals()** asserts that layout block parent is not equal to expected one
    - **string $blockName** block name in layout
    - **string $parentBlockName** parent block name in layout
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockAfter()** asserts that layout block is placed after expected one
    - **string $blockName** block name in layout
    - **string $after** block name in layout, after which it should be placed
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockAfterAll()** asserts that layout block is placed after expected ones
    - **string $blockName** block name in layout

- **array $after** list of block names in layout, after which it should be placed
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockBefore()** asserts that layout block is placed before expected one
    - **string $blockName** block name in layout
    - **string $before** block name in layout, before which it should be placed
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockBeforeAll()** asserts that layout block is placed before expected ones
    - **string $blockName** block name in layout
    - **array $before** list of block names in layout, before which it should be placed
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockRootLevel()** asserts layout block is on the root rendering level
    - **string $blockName** block name in layout
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockNotRootLevel()** asserts layout block is not on the root rendering level
    - **string $blockName** block name in layout
    - **string $message** custom assertion message [optional]

Layout Block Action

These methods useful for testing **<action method=””/>** construction in layout files.

- **assertLayoutBlockActionInvoked()** asserts that an action was invoked for the block
    - **string $blockName** block name in layout
    - **string $method** method that action should call
    - **string $message** custom assertion message [optional]
    - **array $arguments** list of method arguments for additional restriction. Can be associative array, in this case array keys will be checked as well [optional]
    - **string $searchType** type of arguments search matching if they specified. Defines logic for calculation of arguments intersection with the action ones. Possible types:
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_OR** at least one argument exists in the action

        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_AND** all arguments exist in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_EXACT** arguments are the same as in action
- **assertLayoutBlockActionNotInvoked()** asserts that an action was not invoked for the block
    - **string $blockName** block name in layout
    - **string $method** method that action should call
    - **string $message** custom assertion message [optional]
    - **array $arguments** list of method arguments for additional restriction. Can be associative array, in this case array keys will be checked as well [optional]
    - **string $searchType** type of arguments search matching if they specified. Defines logic for calculation of arguments intersection with the action ones. Possible types:
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_OR** at least one argument exists in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_AND** all arguments exist in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_EXACT** arguments are the same as in action
- **assertLayoutBlockActionInvokedAtLeast()** asserts that an action was invoked for the block at least expected number of times
    - **string $blockName** block name in layout
    - **string $method** method that action should call
    - **integer $invokationCount** minimal number of action invoke
    - **string $message** custom assertion message [optional]
    - **array $arguments** list of method arguments for additional restriction. Can be associative array, in this case array keys will be checked as well [optional]
    - **string $searchType** type of arguments search matching if they specified. Defines logic for calculation of arguments intersection with the action ones. Possible types:
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_OR** at least one argument exists in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_AND** all arguments exist in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_EXACT** arguments are

the same as in action

- **assertLayoutBlockActionInvokedExactly()** asserts that an action was invoked for the block exactly expected number of times
    - **string $blockName** block name in layout
    - **string $method** method that action should call
    - **integer $invokationCount** expected number of action invoke
    - **string $message** custom assertion message [optional]
    - **array $arguments** list of method arguments for additional restriction. Can be associative array, in this case array keys will be checked as well [optional]
    - **string $searchType** type of arguments search matching if they specified. Defines logic for calculation of arguments intersection with the action ones. Possible types:
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_OR** at least one argument exists in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_AND** all arguments exist in the action
        - **EcomDev_PHPUnit_Model_Layout::SEARCH_TYPE_EXACT** arguments are the same as in action

Layout Block Properties

For testing your block data and that it is properly set, you can use the following methods. They are using getters for retrieving of the block data.

- **assertLayoutBlockProperty()** asserts that block property value is evaluated by specified constraint
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for block property value assertion
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNot()** asserts that block property value is not evaluated by specified constraint
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **PHPUnit_Framework_Constraint $constraint** constraint instance that will be used for block property value assertion

    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyEquals()** asserts that block property value is equal to expected one
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **mixed $expectedValue** expected property value
    - **string $message** custom assertion message [optional]
    - **float $delta** delta for comparing float values [optional]
    - **integer $maxDepth** the maximum depth for comparing nested level structures (multidimensional arrays, objects, etc) [optional]
    - **boolean $canonicalize** indicates if it is required to sort compared arrays or replace strings line endings for comparing similar values [optional]
    - **boolean $ignoreCase** indicates if strings should be compared in case insensitive mode [optional]
- **assertLayoutBlockPropertyNotEquals()** asserts that block property value is equal to expected one
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **mixed $expectedValue** expected property value
    - **string $message** custom assertion message [optional]
    - **float $delta** delta for comparing float values [optional]
    - **integer $maxDepth** the maximum depth for comparing nested level structures (multidimensional arrays, objects, etc) [optional]
    - **boolean $canonicalize** indicates if it is required to sort compared arrays or replace strings line endings for comparing similar values [optional]
    - **boolean $ignoreCase** indicates if strings should be compared in case insensitive mode [optional]
- **assertLayoutBlockPropertySame()** asserts that block property value is the same as expected one
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **mixed $expectedValue** expected property value

    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNotSame()** asserts that block property value is not the same as expected one
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **mixed $expectedValue** expected property value
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyType()** asserts that block property is equal to expected php internal type
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $type** internal php type
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNotType()** asserts that block property is not equal to expected php internal type
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $type** internal php type
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyInstanceOf()** asserts that layout block property is an instance of expected class name
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $expectedClassName** class name for assertion
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNotInstanceOf()** asserts that layout block property is not an instance of expected class name
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $expectedClassName** class name for assertion
    - **string $message** custom assertion message [optional]

- **assertLayoutBlockPropertyEmpty()** asserts that layout block property value is empty
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNotEmpty()** asserts that layout block property value is not empty
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNull()** asserts that layout block property value is null
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $message** custom assertion message [optional]
- **assertLayoutBlockPropertyNotNull()** asserts that layout block property value is not null
    - **string $blockName** block name in layout
    - **string $propertyName** block property name, like in **getData()** method call
    - **string $message** custom assertion message [optional]