QA-Tools Documentation

Release 1.0.0

Alexander Obuhovich

Oct 01, 2017

Contents

1 Getting Started **3**
- 1.1 Installation . . . 3
- 1.2 Configuration . . . 3
- 1.3 Connecting to Behat . . . 4
- 1.4 Connecting to PHPUnit . . . 5

2 Glossary **7**
- 2.1 Page Element . . . 7
- 2.2 Page Object . . . 7
- 2.3 Element Proxy . . . 7
- 2.4 Annotation . . . 7
- 2.5 Page Factory . . . 7
- 2.6 Page Locator . . . 8

3 Annotations **9**
- 3.1 @find-by . . . 9
- 3.2 @page-url . . . 12
- 3.3 @match-url-exact . . . 14
- 3.4 @match-url-regexp . . . 14
- 3.5 @match-url-component . . . 15
- 3.6 @timeout . . . 15
- 3.7 @element-name . . . 16
- 3.8 @bem . . . 17

4 Best Practices **19**

5 How-To **21**
- 5.1 Using DefaultPageLocator . . . 21
- 5.2 Defining a custom locator . . . 21

...

CHAPTER 1

Getting Started

...

Installation

The library can be installed easily via Composer.

1. Define the dependencies in your `composer.json` file:

```
{
    "require": {
        "qa-tools/qa-tools": "~1.0"
    }
}
```

2. Install/update your vendors:

```
$ curl http://getcomposer.org/installer | php
$ php composer.phar install
```

To verify successful installation look for the `qa-tools/qa-tools` folder within `/vendor/` folder of your project.

Configuration

The library can be optionally configured using following approach:

```
<?php
use Behat\Mink\Driver\Selenium2Driver;
use Behat\Mink\Session;
use QATools\QATools\PageObject\Config\Config;
use QATools\QATools\PageObject\Container;
use QATools\QATools\PageObject\PageFactory;
```

```php

// 1. Obtain/create Mink's session object:
$session = new Session(new Selenium2Driver());

// 2a. Either configure page factory via Config class:
$config = new Config(array(
    'base_url' => 'http://www.example.com',
));
$page_factory = new PageFactory($session, $config);

// 2b. Or configure page factory via dependency injection container:
$container = new Container();
$container['config_options'] = array(
    'base_url' => 'http://www.example.com',
);
$page_factory = new PageFactory($session, $container);
```

Then created `PageFactory` class instance can be used to spawn `Page` class instances at will.

Note: If several Mink's sessions are used (e.g. for different browsers), then separate `PageFactory` class instance needs to be created for each of them. Configuration setting can be shared across different `PageFactory` class instances, when **same container** is used to create them.

Configuration Options

The following configuration options are available:

- `base_url` - allows to specify Base URL to be used to transform all relative urls from *@page-url annotations* into absolute urls.
- `page_namespace_prefix` - array of namespaces in which the *DefaultPageLocator* will search for page classes. Defaults to:

```
array('\\')
```

- `page_url_matchers` - array of classes, that are used to detect if given `Page` is currently opened. Defaults to:

```
array(
    '\\QATools\\QATools\\PageObject\\PageUrlMatcher\\ExactPageUrlMatcher',
    '\\QATools\\QATools\\PageObject\\PageUrlMatcher\\RegexpPageUrlMatcher',
    '\\QATools\\QATools\\PageObject\\PageUrlMatcher\\ComponentPageUrlMatcher',
)
```

If port is specified as part of *base_url* then it will be used in every built url unless specified explicitly in the *@page-url* annotation.

Connecting to Behat

Note: Throughout this tutorial it's assumed that working Behat with MinkExtension is configured and connected to a project that needs to be tested.

To use library with Behat you are required to also install https://github.com/qa-tools/behat-extension.

Connecting to PHPUnit

Note: Throughout this tutorial it's assumed that working PHPUnit is configured and connected to a project that needs to be tested.

To use library with PHPUnit you are required to also install https://github.com/qa-tools/phpunit-extension.

Important: **TODO:** Write about the obtaining Mink session need for PageFactory and that it can be easily done using PHPUnit-Mink.

Connecting PHPUnit-Mink

...

Creating Test Case File

...

CHAPTER 2

Glossary

Important: **TODO:** Explain each term and how they are used to organize overall experience within a library.

Page Element

...

Page Object

...

Element Proxy

...

Annotation

...

Page Factory

...

Page Locator

The page locator, which is used by the page factory, transforms a page name into a fully qualified class name. For usage see *How-To section*.

CHAPTER 3

Annotations

Annotations are widely used to provide access to the library's functionality without a need to write any code.

@find-by

This annotation is used to specify means, by which elements can be found on a page. For example:

```
<?php
use QATools\QATools\PageObject\Element\WebElement;
use QATools\QATools\PageObject\Page;

/**
 * @find-by('id' => 'breadcrumbs')
 * @timeout(2)
 * @element-name('Breadcrumbs Control')
 */
class Breadcrumbs extends WebElement
{

}

class HomePage extends Page
{
    /**
     * @var Breadcrumbs
     */
    protected $breadcrumbsDefault;
}

class AboutPage extends Page
{
    /**
     * @var Breadcrumbs
     * @find-by('css' => 'nav.breadcrumbs')
```

```
28      * @timeout(3)
29      * @element-name('Small Breadcrumbs Control')
30      */
31     protected $breadcrumbsOverride;
32 }
```

When defining an element (line 10) it's possible to specify default `@find-by` annotation (line 6), that will be used when none was specified on a page property (line 17-19). If `@find-by` annotation was specified on a page property (line 27), then it will override any default annotation, that might have been specified on the element's class.

Note: Apart then in page classes it's also possible to have properties with `@find-by` annotations on element sub-classes, that allow presence of sub-elements (e.g. `AbstractElementContainer`).

Locating elements by Class Name

```
@find-by('className' => 'example-class')
@find-by('how' => 'className', 'using' => 'example-class')
```

The above annotations would find **any elements**, which contains class `example-class` among classes assigned to them. For example:

```
<div class="example-class"></div>
<div class="class-a example-class"></div>
<div class="example-class class-b"></div>
<div class="class-a example-class class-b"></div>
```

Locating elements by CSS Selector

```
@find-by('css' => '#test > input.example[type="hidden"]')
@find-by('how' => 'css', 'using' => '#test > input.example[type="hidden"]')
```

The above annotations would find elements, which matches `#test > input.example[type="hidden"]` CSS selector. For example:

```
<input type="hidden" class="example"/>
```

Locating elements by ID Attribute

```
@find-by('id' => 'test')
@find-by('how' => 'id', 'using' => 'test')
```

The above annotations would find **any elements** which `id` attribute matches `test`. For example:

```
<div id="test"></div>
<input id="test"/>
<label id="test"></label>
```

Locating elements by NAME Attribute

```
@find-by('name' => 'test')
@find-by('how' => 'name', 'using' => 'test')
```

The above annotations would find **any elements** which `name` attribute matches `test`. For example:

```
<input name="test"/>
<select name="test"></select>
```

Locating elements by Tag Name

```
@find-by('tagName' => 'section')
@find-by('how' => 'tagName', 'using' => 'section')
```

The above annotations would find **any elements** which tag name matches `section`. For example:

```
<section name="first">test</section>
<section name="second">test</section>
<section></section>
```

Locating links by Full Text

```
@find-by('linkText' => 'the link')
@find-by('how' => 'linkText', 'using' => 'the link')
```

The above annotations would find **any links** (the `A` tag with `href` attribute) which visible text (not including HTML markup) is `the link`. For example:

```
<a href="#">the link</a>
<a href="http://www.google.com">the link</a>
<a href="http://www.google.com">the <strong>link</strong></a>
```

Locating links by Partial Text

```
@find-by('partialLinkText' => 'the link')
@find-by('how' => 'partialLinkText', 'using' => 'the link')
```

The above annotations would find **any links** (the `A` tag with `href` attribute) which visible text (not including HTML markup) contains `the link`. For example:

```
<a href="#">the link</a>
<a href="http://www.google.com">this is the link</a>
<a href="http://www.google.com">the <strong>link</strong> is here</a>
```

Locating elements by XPATH

```
@find-by('xpath' => 'a[@href = "#"]')
@find-by('how' => 'xpath', 'using' => 'a[@href = "#"]')
```

The above annotations would find `A` tags which `href` attribute value matches `#`. For example:

```
<a href="#">the link</a>
<a href="#">the link is here</a>
```

@page-url

This annotation is used to specify URL associated with each page class. For example:

```
<?php
use QATools\QATools\PageObject\Page;

/**
 * @page-url('http://www.example.com/products/shoes.html?color=red')
 * @match-url-exact('http://www.example.com/products/shoes.html?color=red')
 */
class NewProductsPage extends Page
{

}
```

There are several parameters to configure the URL:

1. `url` - specifies absolute or relative URL to the page (mandatory)
2. `params` - specifies additional URL parameters in associative array format (defaults to `array()`)
3. `secure` - specifies if secure connection (the `https://` protocol) needs to be used (defaults to `null`)

Warning: If some/all parameter names are omitted (as seen in above example), then parameter order must be preserved.

Warning: This annotation should only be used on `Page` classes and its subclasses.

Below is an example of annotation, where `url` parameter name is omitted, but `secure` parameter name is not:

```
@page-url('http://www.example.com/products/shoes.html', 'secure' => false)
```

Parameters in the URL

Parameters in the url can be specified using 3 ways:

- after question mark (`?`) in `url` annotation parameter (first)
- in `params` annotation parameter via associative array (second)
- as a combination of both methods from above

All of the following annotations will produce exactly same URL:

```
@page-url('http://www.example.com/products/shoes.html?color=red&size=42')
@page-url('http://www.example.com/products/shoes.html', array('color' => 'red', 'size
↪' => 42))
@page-url('http://www.example.com/products/shoes.html?color=red', array('size' => 42))
```

Same annotations can also be written in long form:

```
@page-url('url' => 'http://www.example.com/products/shoes.html?color=red&size=42')
@page-url('url' => 'http://www.example.com/products/shoes.html', 'params' => array(
↪'color' => 'red', 'size' => 42))
@page-url('url' => 'http://www.example.com/products/shoes.html?color=red', 'params' =>
↪ array('size' => 42))
```

Note: If same URL parameter is specified in both `url` and `params` annotation parameters, then it's value from `params` takes precedence.

Relative URLs

Specifying absolute URLs in each of created Page classes introduces fair amount of code duplication. This becomes even larger problem, when need arises to use Page classes in parallel on 2+ different domains (e.g. local development environment and Continuous Integration server).

After setting `base_url` *configuration option* it will be possible to use relative URLs **along side** (can use both at same time for different pages) with absolute ones:

```
@page-url('products/shoes.html?color=red&size=42')
@page-url('products/shoes.html', array('color' => 'red', 'size' => 42))
@page-url('products/shoes.html?color=red', array('size' => 42))

@page-url('url' => 'products/shoes.html?color=red&size=42')
@page-url('url' => 'products/shoes.html', 'params' => array('color' => 'red', 'size'␣
↪=> 42))
@page-url('url' => 'products/shoes.html?color=red', 'params' => array('size' => 42))
```

Secure/unsecure URLs

If `secure` annotation parameter not specified, then protocol from following sources is used:

- for absolute urls: the `url` parameter of `@page-url` annotation
- for relative urls: `base_url` configuration option

By specifying `secure` annotation parameter (3rd) value it's possible to force secure/unsecure connection. For example:

```
// force secure:
@page-url('products/shoes.html', true)
@page-url('url' => 'products/shoes.html', 'secure' => true)
@page-url('url' => 'http://www.example.com/products/shoes.html', 'secure' => true)

// force non-secure:
@page-url('products/shoes.html', false)
@page-url('url' => 'products/shoes.html', 'secure' => false)
@page-url('url' => 'https://www.example.com/products/shoes.html', 'secure' => false)
```

Custom ports

Like for *base_url* it is possible to include a port in an absolute URL:

```
@page-url('http://www.example.com:8080/products/shoes.html')
```

URL parameter unmasking

It is possible to make `url` parameter of `@page-url` annotation more dynamic (currently it's pretty static) though usage of url masks. The `url mask` is a query string parameter name wrapped within `{` and `}` like so: `{parameter_name}`. When a query string parameter is encountered in the url in such a form, then instead of being added to the query string of built url it would be unmasked (substituted) in the main url part itself.

```
@page-url('products/{product-name}.html', 'params' => array('product-name' => 'shoes
↪'))
```

It doesn't look too powerful right now, but considering that params would be supplied later in `Page::open` method call it would be a major time saver for SEO url building.

Note: Every part of url, except anchor and query string itself can be unmasked in such a way.

@match-url-exact

This annotation allows to check if a specific page is open by comparing the specified full URL against the currently opened URL.

```
1  <?php
2  use QATools\QATools\PageObject\Page;
3
4  /**
5   * @page-url('http://www.example.com/products/shoes.html?color=red')
6   * @match-url-exact('http://www.example.com/products/shoes.html?color=red')
7   */
8  class NewProductsPage extends Page
9  {
10
11 }
```

Warning: This annotation should only be used on `Page` classes and its subclasses.

@match-url-regexp

This annotation allows to check if a specific page is opened using a regular expression against the currently opened URL.

```
@match-url-regexp('/shoes\.html\?color=.+?$/')
@match-url-regexp('regexp' => '/shoes\.html\?color=.+?$/')
```

Warning: This annotation should only be used on `Page` classes and its subclasses.

@match-url-component

This annotation allows to check if a specific page is opened by comparing different components of the URL against the currently opened URL.

There are several parameters to configure the matching, similar to `@page-url`:

1. `path` - specifies the path of an URL
2. `params` - specifies parameters in associative array format
3. `secure` - specifies if secure connection is used
4. `anchor` - specifies the fragment/anchor of an URL
5. `host` - specifies the host of an URL
6. `port` - specifies the port of an URL
7. `user` - specifies the user of an URL
8. `pass` - specifies the password of an URL

```
// matches one of the url components
@match-url-component('path' => '/products/shoes.html')
@match-url-component('secure' => false)
@match-url-component('params' => array('color' => 'red'))
@match-url-component('anchor' => 'fragment')
@match-url-component('host' => 'domain.tld')
@match-url-component('port' => 80)
@match-url-component('user' => 'username')
@match-url-component('pass' => 'password')

// matches several of the url components
@match-url-component('path' => '/products/shoes.html', 'params' => array('color' =>
↪'red'), 'secure' => false, 'host' => 'domain.tld', 'port' => 80, 'user' => 'username
↪', 'pass' => 'password')
```

Warning: This annotation should only be used on `Page` classes and its subclasses.

Note: When several `@match-url-...` annotations are present, then they're checked until first match.

@timeout

This annotation is used to specify maximum waiting time (in seconds), after which search attempt for an element, that is absent on a page, will be considered as *failure* and exception will the thrown.

Note: When `@timeout` annotation is not specified, then search attempt will be considered as *failure* immediately after element won't be found on a page.

For example:

```
1  <?php
2  use QATools\QATools\PageObject\Element\WebElement;
3  use QATools\QATools\PageObject\Page;
4
5  /**
6   * @find-by('id' => 'breadcrumbs')
7   * @timeout(2)
8   * @element-name('Breadcrumbs Control')
9   */
10 class Breadcrumbs extends WebElement
11 {
12
13 }
14
15 class HomePage extends Page
16 {
17     /**
18      * @var Breadcrumbs
19      */
20     protected $breadcrumbsDefault;
21 }
22
23 class AboutPage extends Page
24 {
25     /**
26      * @var Breadcrumbs
27      * @find-by('css' => 'nav.breadcrumbs')
28      * @timeout(3)
29      * @element-name('Small Breadcrumbs Control')
30      */
31     protected $breadcrumbsOverride;
32 }
```

When defining an element (line 10) it's possible to specify default `@timeout` annotation (line 7), that will be used when none was specified on a page property (line 17-19). If `@timeout` annotation was specified on a page property (line 28), then it will override any default annotation, that might have been specified on the element's class.

Note: This annotation can be particularly useful, when dealing with AJAX requests in which element in question would be only present on a page after AJAX request is over.

@element-name

This annotation is used to specify human readable name (or semantic meaning) for a particular usage of the element. Later name, which is set with the help of this annotation will be used in all error messages related to particular element usage on that page.

For example:

```
1  <?php
2  use QATools\QATools\PageObject\Element\WebElement;
3  use QATools\QATools\PageObject\Page;
4
5  /**
6   * @find-by('id' => 'breadcrumbs')
7   * @timeout(2)
8   * @element-name('Breadcrumbs Control')
9   */
10 class Breadcrumbs extends WebElement
11 {
12
13 }
14
15 class HomePage extends Page
16 {
17     /**
18      * @var Breadcrumbs
19      */
20     protected $breadcrumbsDefault;
21 }
22
23 class AboutPage extends Page
24 {
25     /**
26      * @var Breadcrumbs
27      * @find-by('css' => 'nav.breadcrumbs')
28      * @timeout(3)
29      * @element-name('Small Breadcrumbs Control')
30      */
31     protected $breadcrumbsOverride;
32 }
```

When defining an element (line 10) it's possible to specify default `@element-name` annotation (line 8), that will be used when none was specified on a page property (line 17-19). If `@element-name` annotation was specified on a page property (line 29), then it will override any default annotation, that might have been specified on the element's class.

Note: If element name is not specified, then page property name, e.g. `HomePage::$breadcrumbsDefault`, would be used instead.

@bem

...

CHAPTER 4

Best Practices

Important: **TODO:** Write about how to use the each element to get most effect from it. Describe various usages (AJAX, non-ajax, combined element usage to get desired effect).

CHAPTER 5

How-To

Using DefaultPageLocator

The default page locator uses the following strategy to locate pages:

1. uppercases the first letter of each word and joining them to a CamelCase like class name
2. prepending configured namespaces defined via *page_namespace_prefix*
3. returning first found class

```
<?php
...
$page_locator = new DefaultPageLocator(array('\\shop\\pages', '\\shop\\account\\pages
↪'));
$page_class = $page_locator->resolvePage('Login Page');
$page = new $page_class($page_factory);
...
?>
```

Depending on existence either `\shop\pages\LoginPage` or `\shop\account\pages\LoginPage` will be returned.

Defining a custom locator

In some cases it might be necessary to build a custom page locator. For example to map page names to specific classes.

```
1 <?php
2 use QATools\QATools\PageObject\Exception\PageFactoryException;
3 use QATools\QATools\PageObject\PageLocator\IPageLocator;
4
5 class MappingPageLocator implements IPageLocator
6 {
```

```
    protected $pages = array(
        'Login Page' => '\\shop\\pages\\LoginPage',
        'Registration Page' => '\\shop\\pages\\LoginPage',
        'Landing Page' => '\\shop\\pages\\LoginPage',
        'Account Overview Page' => '\\shop\\account\\pages\\AccountPage',
    );

    public function resolvePage($name)
    {
        if ( !isset($this->pages[$name]) ) {
            throw new PageFactoryException(
                'Couldn\'t locate ' . $name,
                PageFactoryException::TYPE_PAGE_CLASS_NOT_FOUND
            );
        }

        return $this->pages[$name];
    }
}
```

Now it is possible to either locate the page manually by its name:

```
<?php
...
$page_locator = new MappingPageLocator();
$registration_page_class = $page_locator->resolvePage('Registration Page');
$registration_page = new $registration_page_class($page_factory);
...
?>
```

or replace default locator with new one during `PageFactory` construction time.

```
<?php
...
$container = new Container();

$container['page_locator'] = function () {
    return new MappingPageLocator();
};

$page_factory = new PageFactory($session, $container);

$registration_page = $page_factory->getPage('Registration Page');
...
?>
```