

# Zend\_Controller Testing Infrastructure

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

## Zend Framework: Zend\_Controller Testing Infrastructure Component Proposal

<b>Proposed Component Name</b>	Zend_Controller Testing Infrastructure
<b>Developer Notes</b>	<a href="http://framework.zend.com/wiki/display/ZFDEV/Zend_Controller+Testing+Infrastructure">http://framework.zend.com/wiki/display/ZFDEV/Zend_Controller Testing Infrastructure</a>
<b>Proposers</b>	Matthew Weier O'Phinney Zend Liaison Ralph Schindler
<b>Zend Liaison</b>	TBD
<b>Revision</b>	1.0 - 27 May 2008: Community Draft. 1.1 - 27 May 2008: Proposal draft for Zend review (wiki revision: 11)

## Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
  - bootstrap() Method
  - reset() Method
  - setUp() Method
  - dispatch(\$uri) Method
- Sessions
- New Assertions
  - assertSelect\*() family
  - assertXPath\*() family
  - assertResponse(), assertRedirect()
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

## 1. Overview

Since I originally undertook the MVC refactoring project in the fall of 2006, I've been touting the fact that you can test ZF MVC projects by utilizing the Request and Response objects; indeed, this is what I actually did to test the Front Controller and Dispatcher. However, there is not currently an easy way to do so; the default request and response objects make it difficult to easily and quickly setup tests, and the methods introduced into the front controller to make it testable are largely undocumented. The only resource developers currently have for this sort of testing is the unit tests themselves.

Testing against the response object is somewhat tedious; you have to be regex guru in many cases, and the testing itself includes many calls to the response object in order to grab the values to test against.

This proposal aims to simplify testing MVC applications by providing tools to allow assertions against the DOM structure of your response, specialized request/response objects that mimic the HTTP environment and also prevent spurious headers and content from being generated, and more.

## 2. References

- PHPUnit
- dojo.query
- Prototype's \$\$()
- Working POC

## 3. Component Requirements, Constraints, and Acceptance Criteria

The following application areas should be testable:

- Response status: HTTP response codes (assume 200, but several methods will set the response code in the response object)
- Response headers: presence of headers and content of headers
- Content:
  - what nodes are present, selected by DOM id and/or CSS class selectors
  - content of selected nodes, both straight string comparisons and by regex

## 4. Dependencies on Other Framework Components

- Zend\_Controller
- Zend\_Exception
- PHPUnit

## 5. Theory of Operation

In most cases, you don't need to go into terribly fine-grained detail, but you need to verify that particular elements or types of elements are present in the response, as well as potentially match small segments of the content. The response status is generally desirable, and often you need to test where an action may redirect.

When selecting nodes in XML or XHTML, the "gold standard" is represented by Prototype's \$\$() function, and Dojo Toolkit's dojo.query() method. Each of these utilize CSS selectors to qualify the nodes to select; CSS selectors allow us to specify specific node types, node ids, as well as the values contained in the 'class' attribute of nodes; additionally, it has a concept of inheritance, to indicate parent/child relations. This kind of specificity can allow a developer to check that *types* of content are present, instead of the *specific* content – which may be variable, due to having unreliable sources (database, for instance).

Additionally, if a developer wishes to use straight XPath, they should be able to do so as well.

I'm proposing several items. First, a component for selecting against DOM artifacts:

- Zend\_Dom\_Query

This class is named to indicate first, its function, and second, an affinity for dojo.query() (since \$\$ is a name that's hard to translate to PHP). This component would be utilized to parse response body content. It would allow querying via CSS selectors or XPath.

Second, a test case for controllers that extends PHPUnit\_Extensions\_Database\_TestCase:

- Zend\_Test\_PHPUnit\_ControllerTestCase (extends PHPUnit\_Extensions\_Database\_TestCase)

This test case would provide scaffolding for bootstrapping the MVC application being tested, methods for dispatching a request and retrieving the

response, and a number of assertions for running against the request.

The assertions include:

- `assertSelect()` (assert that nodes specified by CSS selectors exist)
- `assertNotSelect()`
- `assertSelectContentContains()` (assert that nodes found by CSS selector contain the text specified)
- `assertNotSelectContentContains()`
- `assertSelectContentRegex()` (assert that nodes found by CSS selector match the pattern specified)
- `assertNotSelectContentRegex()`
- `assertSelectCount()` (assert that exactly the number of nodes found by CSS selector are found)
- `assertNotSelectCount()`
- `assertSelectCountMin()` (assert that at least the specified number of nodes are found by CSS selector)
- `assertSelectCountMax()` (assert that at most the specified number of nodes are found by CSS selector)
- `assertXPath()` (assert that nodes specified by XPaths exist)
- `assertNotXPath()`
- `assertXPathContentContains()` (assert that nodes found by XPath contain the text specified)
- `assertNotXPathContentContains()`
- `assertXPathContentRegex()` (assert that nodes found by XPath match the pattern specified)
- `assertNotXPathContentRegex()`
- `assertXPathCount()` (assert that exactly the number of nodes found by XPath are found)
- `assertNotXPathCount()`
- `assertXPathCountMin()` (assert that at least the specified number of nodes are found by XPath)
- `assertXPathCountMax()` (assert that at most the specified number of nodes are found by XPath)
- `assertResponseCode()` (assert that the given response code was given)
- `assertNotResponseCode()`
- `assertHeader()` (assert that a given header type was provided)
- `assertNotHeader()`
- `assertHeaderContains()` (assert that a given header type was provided and contains the text specified)
- `assertNotHeaderContains()`
- `assertHeaderRegex()` (assert that a given header type was provided and matches the pattern specified)
- `assertNotHeaderRegex()`
- `assertRedirect()` (assert that the response is a redirect)
- `assertNotRedirect()` (assert that the response is a redirect)
- `assertRedirectTo()` (assert that the response redirects to the given url)
- `assertNotRedirectTo()`
- `assertRedirectRegex()` (assert that the response redirects to a url matching the given pattern)
- `assertNotRedirectRegex()`
- `assert(Not)AttributeExists($domNode, $attr)`
- `assert(Not)AttributeEquals($domNode, $attr, $value)`
- `assertDataSetInsert($table)`
- `assertDataSetDelete($table, $id)`
- `assertDataSetUpdated($table, $id)`

Finally, custom request/response objects would also be created to facilitate testing (in particular, adding accessors to set request data), with corresponding interfaces for the functionality required by testing. This latter would facilitate custom request/response objects provided by the developer.

- `Zend_Controller_Request_HttpTestCase`
- `Zend_Controller_Response_HttpTestCase`

## **bootstrap() Method**

By default, checks if `$bootstrapFile` is set, and if so, includes that file.

Override the method to provide specific functionality.

If using a bootstrap file, the file should not dispatch. Additionally, users should be aware that custom request and response objects will be set in the front controller if none are set in the `TestCase`.

## **reset() Method**

Reset's object state of front controller, and clears request/response objects.

## setUp() Method

By default, calls reset() and bootstrap(). When overriding, should call parent::setUp().

## dispatch(\$uri) Method

Sets \$uri in request object, and then dispatches front controller.

Sets returnResponse() in front controller to ensure response is returned.

## Sessions

May need to create a Zend\_Session replacement that can be used within tests. Must be compatible with Zend\_Session\_Namespace. Will be an object store to allow you to check the contents of the session to see if data has been set.

## New Assertions

### assertSelect\*() family

Basically, these operates similar to Prototype's \$\$() function; you pass them CSS selector notation, which allows for ids, classes, and inheritance. Based on the assertion type and arguments you pass, you assert different things:

- assertSelect('#foo') - foo id exists
- assertNotSelect('#foo') - foo id does not exist
- assertSelectContentContains('#foo', 'bar') - content of foo contains 'bar'
- assertNotSelectContentContains('#foo', 'bar') - content of foo does not contain 'bar'
- assertSelectContentRegex('#foo', '/bar/i') - regex match on content of foo
- assertNotSelectContentRegex('#foo', '/bar/i') - negative regex match on content of foo
- assertSelectCountMin('td#foo>tr', 3) - at least three rows in the foo table
- assertSelectCountMax('td#foo>tr', 3) - no more than three rows in the foo table

All assertions would accept an optional extra argument, a \$message to return on failure.

### assertXPath\*() family

All assertSelect\* assertions would have corresponding assertXPath assertions that accept an XPath expression as an argument.

### assertResponse(), assertRedirect()

Tests various properties of the response:

- assertResponseCode(200) - 200 OK response code
- assertResponseCode(404) - assert page not found
- assertNotResponseCode(404) - assert NOT page not found
- assertResponseCode(500) - assert application error
- assertHeader('X-App-Id') - X-App-Id header in response
- assertNotHeader('X-App-Id') - X-App-Id header not in response
- assertHeaderContains('X-App-Id', 'bar') - X-App-Id header contains 'bar'
- assertHeaderRegex('X-App-Id', '/bar/i') - regex match on X-App-Id header
- assertRedirect() - response is a redirect
- assertNotRedirect() - response is not a redirect
- assertRedirectTo('/foo') - response redirects to /foo
- assertNotRedirectTo('/foo') - response does not redirect to /foo
- assertRedirectRegex('|^/foo|') - regex match on response redirect
- assertNotRedirectRegex('|^/foo|') - negative regex match on response redirect

## 6. Milestones / Tasks

- Milestone 1: [DONE] Submit proposal for review
- Milestone 2: Complete working tests and code for implementation
- Milestone 3: Documentation, use cases, and tutorials

## 7. Class Index

- Zend\_Controller\_Request\_HttpTestCase
- Zend\_Controller\_Response\_HttpTestCase
- Zend\_Dom\_Exception
- Zend\_Dom\_Query
- Zend\_Dom\_Query\_Css2XPath
- Zend\_Dom\_Query\_Result
- Zend\_Test\_PHPUnit\_ControllerTestCase
- Zend\_Test\_PHPUnit\_Constraint\_Exception
- Zend\_Test\_PHPUnit\_Constraint\_DomQuery
- Zend\_Test\_PHPUnit\_Constraint\_Redirect
- Zend\_Test\_PHPUnit\_Constraint\_ResponseHeader

## 8. Use Cases

UC-01

```
class FooControllerTest extends Zend_Test_PHPUnit_ControllerTestCase
{
    public $bootstrap = dirname(__FILE__) . '/../bootstrap.php';

    public function testArchivesRouteShouldContainNabble()
    {
        $this->dispatch('/archives');
        $this->assertResponse(200);
        $this->assertSelect('#nabble');
    }

    public function testNabbleInArchivesShouldContainScript()
    {
        $this->testArchivesRouteShouldContainNabble();
        $this->assertSelectContains('#nabble', '<script');
    }

    public function testAddPersonShouldRedirect()
    {
        $this->getRequest()->setPost($this->newPersonData);
        $this->dispatch('/person/add');
        $this->assertRedirect();
    }
}
```

## 9. Class Skeletons

```
]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>
```