Des McCarter (@BJSS)

des.mccarter@bjss.com

Abstract

A guide for creating robust and reliable test frameworks based on simple testing and development principles within an enterprise environment

MAXWELL test framework REFERENCE GUIDE

A general guide to web based automation testing

Contents

[Introduction 2](#_Toc489014931)

[A typical framework implementation approach 3](#_Toc489014932)

[Tools QA page factory – the *traditional* page object representation approach 3](#_Toc489014933)

[Using page factories to manage page interactions 3](#_Toc489014934)

[Framework approach 5](#_Toc489014935)

[General page factory/management approach 5](#_Toc489014936)

[Page management in Maxwell 5](#_Toc489014937)

[Example page expression 6](#_Toc489014938)

[Using page expressions 7](#_Toc489014939)

[Switching between browsers 8](#_Toc489014940)

[Test flow management 9](#_Toc489014941)

Named in honour of James Clerk Maxwell



# Introduction

This document should be used both as a technical reference guide to using the Maxwell test framework and also as a general approach to developing an enterprise test framework similar to Maxwell. Information here includes the pitfalls found when developing tests around a highly volatile application such as K2 ([www.k2.org](http://www.k2.org)).

K2 is a .NET application framework which allows developers to develop an application with minimal code. Maxwell was initially developed against an application required by HS2 and was found problematic to test given its intolerant nature both from a functional and performance perspective (both in manual and automated testing). This document includes a mention of these pitfalls as well as their solutions.

The Maxwell framework was initially developed to functionally test a product developed at HS2 by BJSS. The core of the framework comprises of two bespoke libraries: *pageengine* and *actionengine*. Pageengine manages the physical interactions between web based tests and the web page’s themselves. Actionengine was developed to manage the execution flow, usability and re-usability of the tests cases / workflow steps / actions.

# A typical framework implementation approach

A typical tactical approach normally taken when developing a web based front end framework is to a. use (or build) a page factory library for retrieving page information (i.e. information on the pages being tested) and b. include test execution flow/steps within code (or scripts).

Another approach (which is poor, clumbersome and extremely difficult to manage) is to manage page interactions DIRECTLY in the test itself along with the test flows. The latter I have seen at various organisations (Waitrose for example) and I will only advise as route NOT to take and must be avoided (especially within an enterprise development environment) at all costs IF you require tests to be stable and manageable.

## The *traditional* page factory approach

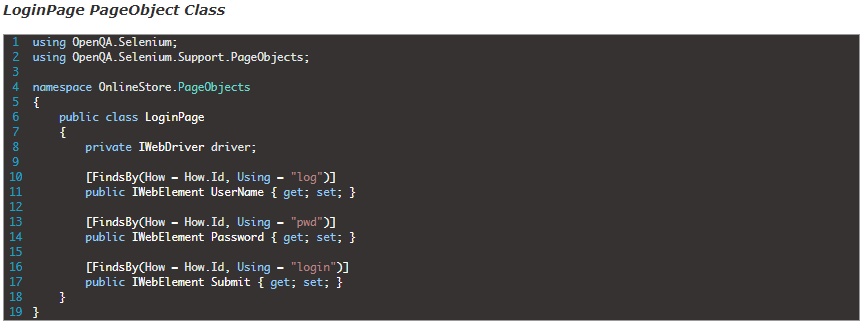
The former is the approach taken in the development of Maxwell but has a slight difference to the traditional page factory implementations. The main difference is the approach taken in how pages (and their elements) are defined. Most page factories define pages in code, where pages are represented by classes. Elements of these pages are represented by properties (or getters/setters) of the class. Using Maxwell, all pages and their elements are defined within an XML file (pages.xml).

## Example of the traditional page factory approach - ToolsQA

Libraries such as the one developed by Tools QA (<http://toolsqa.com/selenium-webdriver/page-object-pattern-model-page-factory/>) use the traditional class approach for defining pages and their elements. This general approach is absolutely fine but most require coders to define and describe pages in code in the form of classes (per page). A page would typically be represented by a class with their buttons, inputs, text boxes etc represented by getters/setters (or properties in .NET).

### Defining pages

Example traditional page representation in code (Tools QA - from <http://toolsqa.com/selenium-webdriver/c-sharp/pagefactory-in-c/>)



The image above illustrates how pages are typically defined and represented. This example is used by ToolsQA, where the page itself (under test) is represented by a class, named LoginPage. This class has properties (getters and setters) which represent each element to be interacted with on the Login page: *login*, *password* and *log*.

### Using page factory definitions (page objects) in tests

Example Tools QA / traditional page object usage

The image below shows how the previously defined page object is used based on the Tools QA page factory. The steps used are as follows:

1. ***Define the driver object (the object which dictates the browser to be used in the test (Firefox in this case).***



1. ***Set the URL property of the driver object to the page under test.***



1. ***Initialise the driver with the page(s) definition by calling InitElements() for each page.***

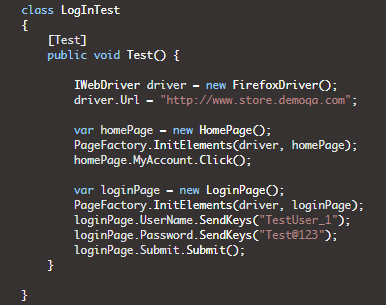


… and then finally …

1. Start interacting (clicking, sending text etc) with the page.



A typical class (in this case, NUNIT based) would look like this:



# Maxwell Framework approach

## Drawbacks to traditional page factory/management approach

Whilst these separation of concerns makes sense (the concerns in this case being page and test execution flow management), in the real world, *HOW* this is implemented is one of the keys to having suitably reliable set of tests. This increasingly becomes important as the number of test cases developed becomes a significant count and even more so if the application being tested is enterprise.

For example, if the xpath location of a certain button has changed then you will need to update the class and rebuild the tests etc to reflect the change. Also, if the xpath location of a certain elements vary between each environment (this has been known to happen) then you will need to branch the tests in source *control PER ENVIRONMENT*!.

## Page management in Maxwell

Pages under test are managed in Maxwell by a. expressing/representing their detail within an XML file and b. using this representation in code. This differs from the typical approach by a. how pages are defined and expressed and b. how pages and drivers are initialised.

Example page expression – representing Expedia.co.uk

The image below shows an example of how pages are defined and expressed in Maxwell. All pages are defined within one XML file. The page under test itself is defined within its own ***Page XML element*** and each page element (i.e. each element on the page which is to be interacted with in the test, e.g. buttons, text boxes etc) are defined as ***Element XML elements*** within the Page XML element.



In the simplest case, each page definition has a name …



… and a URL associated with that name …



The page *Name* element is used within the test to retrieve an instance of that page whereas the URL is used by pageengine to navigate to that page.

### Using page expressions

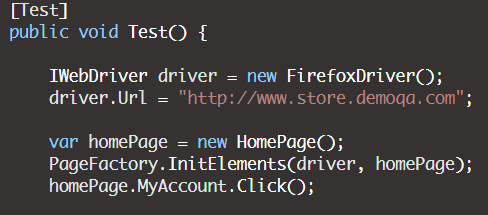
Page expressions are used by a. first getting an instance of this page by calling the OpenPage method (which also implicitly navigates to that page using the selected browser) b. applying actions to the page elements, such as clicking them, reading text, sending text etc. An example of this is shown below using the page ExpediaPage and ExpediaDeparturePage expressions shown earlier:



Note also that any Selenium based functionality is extracted out from the test. For a test you simply want to a. open the page and b. interact with its elements. This is opposed to the normal approach which is to define additional information, such as declaring an instance of the web driver object and initialising an instance of the defined page against it (along with declaring the page URL):

Example of the *“normal”* approach of interacting with

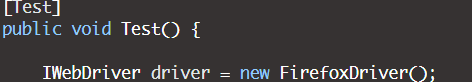
page elements using a traditional page factory



### Switching between browsers

Using the traditional approach, browser drivers are typically defined and initialised within the test itself:

Defining browser (selenium driver) information using typical approach



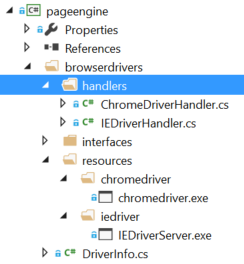
So for every test, in theory, a driver instance will need to be created. In the Maxwell framework browsers are defined (and pointed to) via the tests App.Config file:

Defining browser (selenium driver) information using Maxwell



In this case, all tests will use the browser defined by the ***driver.wrapper.class key*** within the applications App.Config file. In the case shown above the IEDriverHandler class is used. This class handles the initialisation of the specified selenium driver, in this case Internet Explorer (reflected by the *IE* bit of the *IEDriverHandler* class name). Driver wrappers are located within the pageengine project under the browserdrivers/handlers folder:

Browser driver handlers – location in the pageengine project



## Test flow management

<write info ere>