# TestPipe

Various musings about the TestWorks project by the project team.

* Charles Bryant – Developer

1/20/2014

I am thinking about forgoing out of the box unit test frameworks and roll my own runner to allow them to run in async parallel with TPL and .NET 4.5 async keyword.

* Collect all features from DLL. The DLL contain test classes generated by SpecFlow and at the moment can only be NUnit or MSTest.
* Collect all scenarios from the feature. The feature is the test class generated by SpecFlow and scenarios are identified by a method attribute.
* Collect all the steps from the step class. The step classes are identified by a class attribute.

There are two ways we can approach running tests:

* Sequentially calling the scenario methods from the feature class and allow the test to call the steps.
* Sequentially calling the step methods for each scenario where we take over the entire test running process.

Our missing is to do this asynchronously and in parallel.

* Do we have to manage the number of threads? I believe we can pass this complexity off to TPL and allow it to optimize threading.
* How do we handle Selenium Grid? We should only have to worry about setting up the desired capabilities and pass it to Selenium Grid and the Grid will manage selecting the nodes to run on.
* How do we handle
  + Return values (results)
  + Timeouts
  + Exceptions
* What do we have to change in the current framework to allow parallel execution? We should look at shared data and functionality for areas that can produce unwanted side effects. TestSession is used by all tests and is a static class that includes a lot of static methods and holds static state.

Once we have the tests collected we iterate them and call them async across as many remote browsers as we have set up. The async task will either return a Result object or we will interpret the browser’s return value into a Result object. Once all of the tests have ran, we iterate the Result objects to produce a report.

9/15/2013

## TestPipe 0.1

Today I decided to take my stalled browser test framework and breathe some life into it since I need to use a test framework at work. Basically, it’s a generic browser test framework abstraction that allows you to swap out browser test engines (e.g. Selenium, Watir…). I know there are projects out there that do this and the world doesn’t need another abstraction causing confusion in the browser test framework space, but it was the only way I could see getting a framework that I could feel comfortable with.

Everything will have a key. I got tired of thinking about what type the key should be in my objects. So, I decided to give everyone a Guid ID and a string Key. If correct these will always match in the event stream for the instance it represents regardless of the state of the instance. So, I can look up and retrieve an instance of an object by ID or Key and the instance can be from any point in time that I have an event for.

The only reason we can’t fully abstract a page is that we inject a browser into the page. The browser is a very specific object and will be different depending on the engine used to drive the browser. We can have a Selenium driver or a Watir driver or any driver that we can implement against the browser API. So, we construct the browser closer to the test runner and we can’t provide a full abstraction that includes the browser because we would take a dependency on a higher level object and that’s a no, no. One we could create some weird hard to debug circular dependency thing and we impose all of the heavy maintenance that goes on in higher levels on our low level page. If we want to add a driver, and we would have to make code changes, risky code changes, to get it done. There are more reasons, but sometimes it’s better to experience a maintenance nightmare to get the lessons engrained, so I won’t bore you with a rant on layer dependencies.

Should a page know how to build its full URL or should an external caller pass it into the page? Unless the caller is the same as the one constructing the page it will mean an extra dependency. If the page constructor builds the URL, it will have to know the page’s virtual path then it can combine it with the base URL to pass into the page. We know we will have a reference to the application object. It may even be the constructor of the page, but a page constructor or factory sounds more intuitive. Do we want to have another object hold the page’s virtual path? Or will we instantiate the page then ask it for its virtual path? That’s another dependency to hold the path or a round trip to the page to get the path.

I think a page factory is warranted because the page object provides the bridge between the framework and the driver. The introduction of the browser object in the specific driver implementations is where the link is made. Getting the page right is very important and extra code to get it right is worth it. But why do we have to use a totally different interface. ICollection<T> is wonderful. What’s wrong with IPage<T> where T is the specific browser driver.

9/15/2013

## Bounded Context

TestPipe is divided into discrete systems that handle specific contexts in the TestPipe application.

### Application

Application – the application under test.

Environment – the application environments.

Pages – the testable pages in the application.

Controls – the testable controls on the page.

### Test Definition

Suite – the test container (e.g. smoke test, acceptance test, regression test…). Suite manages what environments, features, scenarios, pages are allowed to run. Example: the suite can ignore or add a scenario based on the environment. Using SpecFlow, we could tag a scenario as @datawrite, @dataread, @datareadwrite. Then in the Prod environment only allow the @dataread scenarios to run.

Application – the application tested in the suite.

Environments – the environments allowed in the test.

Feature – the features to be tested in the suite. A feature maps to a test class.

Scenario – the scenarios to be tested in a feature. A scenario maps to a test method.

Pages – the pages to be tested in a scenario.

Step – the steps to take on a page control in a scenario. A step maps to a SpecFlow step class method.

Assert – the assumption to make on a scenario “Then” step (e.g. page == somepage, page.control.text = some text…)

### Test Run Data

Data – the data to inject into the test suite. The application, environment, browser

Suite – the suite data

Features – the suite feature data

Scenarios – the feature scenario data

Step – the scenario step data

Assert – the left side value or expected outcome of the “Then” step assumption.

### Test Run Session

Application – the application under test.

Environment – the application environment to run the test against.

Browser – the browser driver and specific browser to run tests in.

Suite

Features

Scenarios

### Test Results

Each result has a start and end time.

Application - the cumulative result of the suite results in the application under test (this may be configured to ignore some failures or give more or less weight to certain failures in calculating the suite result).

Suite – the cumulative result of the features results in the suite.

Feature – the cumulative result of the scenarios results in the feature.

Scenario – the result of the step assert in the scenario.

Step – the result of the step run.

Assert – the result of the expected versus actual outcome of the “Then” step assumption.

TestPipe.App

* Create App
* Create App Environment
* Change App Environment
* Create App Page
* Change App Page
* Create App Page Control
* Change App Page Control

Control

Because the TestPipe Controller has total knowledge of your tests it is able to reconfigure tests for various requirements. If you need to change drivers to perform a different test, TestPipe can do that. If you need to run a different browser, TestPipe can do that. If you need to run against multiple applications at multiple endpoints, TestPipe can do that. If you need to test web services, JavaScript, mobile…TestPipe can do that.

Next we just need to load a dll, instantiate a class, and run it. Oh wait, the command line already does that and we have PowerShell. We also have reflection and dynamic compilation through the CLR SDK. Before we run a dll we have to match the checksum with the one we have stored when the dll was registered. Registering dll can only happen behind closed doors. It is not an internet thing. Can we tell Norton to scan a DLL through the command line?

Did you know TestPipe can also do custom reporting and is backed by an Event Sourced data source? Maybe, maybe not. Event Sourcing would give a wonderful set of data. Why wouldn’t you not also Event Source analytics. In the end Event Sourcing is logging and being able to project or replay past events within a future context.

Copied from <https://github.com/remi/Mara>

## Description

Mara aims to simplify the process of integration testing .NET web applications, such as ASP.NET, ASP.NET MVC, or WCF. Mara simulates how a real user would interact with a web application. It is agnostic about the driver running your tests and currently comes bundled with Selenium and System.Net.WebClient support built in. It is also agnostic about servers that can automatically server your application and currently comes bundles with Cassini and XSP support built in.

Mara is inspired by [Capybara](https://github.com/jnicklas/capybara), a similar tool written in Ruby.

## HERE BE DRAGONS

Mara is insanely ALPHA right now. I recommend you give me a few weeks to polish it before you start using it or contributing. The API may go through some major changes so ... just be patient. If you're a hacker and you want to check out the code and try to get it working, be my guest!

## Install

We will make Mara available as a [NuGet](http://nuget.codeplex.com/) package. For now, you can clone the solution and build it yourself if youreally want to try it out.

Mara currently works in [Mono](http://www.mono-project.com/) (2.8 required for HtmlUnit with WebDriver) and MS .NET 3.5. Keeping Mara working in [Mono](http://www.mono-project.com/) is and will always be a high priority!

## WebClient

The default driver uses System.Net.WebClient to send requests and [HtmlAgilityPack](http://htmlagilitypack.codeplex.com/) to run XPath queries against the resulting HTML. It does NOT support JavaScript. Because it doesn't execute JavaScript or automate a browser, it is quite fast.

Very shortly, Mara will make it easy to switch between drivers in your tests so you can use WebClient for some of your tests that don't require JavaScript but use WebDriver (or WatiN) for tests that do require JavaScript.

## Selenium 2.0 (WebDriver)

[Selenium WebDriver](http://code.google.com/p/selenium/) is currently the only built in driver that supports JavaScript. We plan on adding [WatiN](http://watin.sourceforge.net/)support in the near future.

I'll add more documentation on how to configure webdriver later ... for now, the most important thing to know is that there are some very useful environment variables that can be used, mainly BROWSER

Let's say that you create an NUnit test (as demonstrated below).

# this will run your tests in WebDriver.DefaultBrowser (Firefox)

nunit-console MyTests.dll

# this will run your tests in Internet Explorer

BROWSER=IE nunit-console MyTests.dll # BASH

SET BROWSER=IE && nunit-console MyTests.dll # BATCH

BROWSER can currently be set to Firefox, Chrome, IE, or HtmlUnit. If HtmlUnit is selected, you should put a copy of selenium-server-standalone.jar (which can be found in the download section of the [Selenium](http://code.google.com/p/selenium/) site) in the current directory. It will automatically be launched.

**NOTE**: HtmlUnit is a Java library so you'll need to have Java installed on your machine if you want to use it. Java is not required to test in the other browsers, just HtmlUnit.

Let's say you have a server that you want to use to run your tests on. Maybe it has Java installed so you want to run your HtmlUnit tests on the remote server and just test with IE locally.

# this will connect to a remote server running selenium-server-standalone and run your tests on that server

# using HtmlUnit

REMOTE=http://remote-server:4444/wd/hub BROWSER=HtmlUnit nunit-console MyTests.dll

**NOTE**: the remote stuff isn't fully fleshed out in Mara. It works great with Selenium, in general, but we'll be sure to make it really easy for you to do stuff like this ... just give us some time :)

## Hello World (the easy way)

If you're using NUnit, we have a Mara.NUnit assembly that will help you get up and running quickly.

using Mara;

namespace MyIntegrationTests {

// This will instantiate a driver/server for you when your test suite starts and will stop them when it stops

[SetUpFixture] public class MaraSetup : MaraSetUpFixture {}

// If your test inherits from MaraTest, you get direct access to all of the driver's methods

// without having to call them on an object. eg. You can call Visit() instead of Driver.Visit()

[TestFixture]

public class TestMyWebSite : MaraTest {

[Test]

public void CanRegister() {

Visit("/");

ClickLink("Register");

FillInFields(new {

Username = "bobsmith",

Email = "bob@smith.com",

Password = "secret",

PasswordConfirmation = "secret"

});

ClickButton("Sign up!");

Assert.That(Find("id('Message')").Text, Is.EqualTo("Successfully Registered")); // XPath is accepted by Find() and All()

}

}

}

That should run!

So ... how does it find your web application? When your tests run, if you haven't explicitly told it where your web application is, it looks in parent directories for a directory containing a Web.config. To manually tell Mara where your ASP.NET application is:

Mara.App = @"C:\Users\me\MyAspNetApp";

It also automatically boots up your application via Cassini (or XSP if running via Mono).

Mara.RunServer = false; // tell Mara not to run your application

Mara.DefaultServerName = "Mara.Servers.Cassini"; // FullName of the class that will be used as a default server

Mara.Server = new MyServer(); // tell Mara to use your server. It needs to use the \*very simply\* IServer interface

Mara.Port = 1234; // set the port that servers use by default

There's a lot more ... the API for some of this stuff will VERY likely change though! You've been warned!

## Hello World (doing it all yourself)

So, you're not using NUnit. Or you don't like the magic. No problem!

using Mara;

namespace MyIntegrationTests {

[SetUpFixture]

public class MyFixture {

static IDriver maraDriver;

[SetUp]

public void Setup() {

// do some configuration (optional)

Mara.App = @"C:\Path\To\My\App\Folder";

// Tell the default Mara.Server to boot up your application (or you could do this manually).

// If you manually start your own server, be sure to set Mara.AppHost to the URL to the root

// of your application, so it can be found by drivers, eg. "http://localhost:1234"

Mara.Server.Start();

// Instantiate a driver. Again, you can do this manually. We'll use the default driver

maraDriver = Mara.Driver; // this instantiates a default driver for you

}

[TearDown]

public void Teardown() {

// Stop the server

Mara.Server.Stop();

// Close the driver

maraDriver.Close();

}

}

[TestFixture]

public class TestMyWebSite {

// do something to make the driver available to your tests

IPage Page { get { return MyFixture.maraPage; }}

[Test]

public void CanRegister() {

Page.Visit("/");

Page.ClickLink("Register");

Page.FillInFields(new {

Username = "bobsmith",

Email = "bob@smith.com",

Password = "secret",

PasswordConfirmation = "secret"

});

Page.ClickButton("Sign up!");

Assert.That(Page.Find("id('Message')").Text, Is.EqualTo("Successfully Registered")); // XPath is accepted by Find() and All()

}

}

}

Assuming I typed that all out correctly, that should get you up and running doing it all yourself.

## The DSL

### Navigating

You can use the Visit method to navigate to other pages:

Visit("/About.aspx");

The visit method only takes a single parameter, the request method is **always** GET.

You can get the current path and url of the browsing session for test assertions:

Assert.True(CurrentPath == "/About.aspx");

Assert.True(CurrentUrl == "http://localhost:8080/About.aspx");

### Clicking links and buttons

You can interact with the webapp by following links and buttons. Mara automatically follows any redirects, and submits forms associated with buttons.

ClickLink("text of link");

ClickButton("value of submit button");

Click("FAQ"); // If there's a Link with the text "FAQ", that is clicked, otherwise tries to ClickButton("FAQ");

### Interacting with forms

Forms are everywhere in webapps, there are a number of tools for interacting with the various form elements:

FillIn("ID or Name of field", "Value");

// Calls FillIn() for each of the key/value pairs passed in

FillInFields(new { Username = "bobsmith", Password = "secret" });

**NOTE**: A number of other form helper methods will be implemented shortly ... there are NOT implemented yet:

Choose("A Radio Button");

Check("A Checkbox");

Uncheck("A Checkbox");

Select("Option", "ID or Name of <select>")

AttachFile("ID or Name of field", "/path/to/image.jpg");

### Querying

Mara has some very simple querying helpers:

Page.HasContent("text");

Page.HasXPath("//h1");

We don't yet support things like querying within scopes, etc, etc. Eventually!

### Finding

You can find elements in order to manipulate them.

var nameField = Find("//input[@name='DogName']");

nameField.Value = "Set the value of this field";

Console.WriteLine("The alt attribute is {0}", nameField["alt"]);

Find("id('MyButton')").Click();

// Or, to find many elements:

foreach (var link in All("//a"))

Console.WriteLine("Link with text {0} goes to {1}", link.Text, link["href"]);

### Scripting

In drivers which support it, you can easily execute JavaScript:

ExecuteScript("$('body').empty()");

For simple expressions, you can return the result of the script. Note that this may break with more complicated expressions:

var result = EvaluateScript("4 + 4");

### Debugging

It can be useful to take a snapshot of the page as it currently is and take a look at it:

SaveAndOpenPage();

## More

There is much, much more ... some of it developed, some not yet.

More documentation and features coming soon!

## License

Mara is released under the MIT license.

## Ignoring Tests

Add property to object model to signify that a test should be ignored and through a TestIgnoredException and log the result as ignored. There will have to be logic baked into BeginStep method to check the ignore property to see if it should be ignored.

We could ignore tests based on state. Only run certain tests in certain environments or browsers or browser drivers.

We could only run tests for a certain release or ticket. If a dev checks in code for a ticket, we could have a service run all tests associated with the ticket. If a branch is ready for release, we could run all tests associated with the release. We don’t have to have an all or nothing only do full regression test suite.

## Reporting Tests

Test Name

Date

Results

Associated Ticket

Associated Release

# **Project Manager Integration**

Currently, we use OnTime for project management. We can use the OnTime API, but we will need authorization to turn the API feature on. With the API, when a user authors a new test they will be able to associate the test with a release and a ticket.

What if a test covers multiple tickets or we want to rerun current tests for a ticket or release?

So, we should be able to assert new and existing tests with releases and tickets. When the test runner reports the suite, feature, and scenario it wants to run we can opt to ignore it or run it. If a test is not included in a suite it isn’t ran. This allows us to ignore features in the test run. If we are running a test suite for a ticket and a scenario is not associated with the ticket, it is ignored.

Biz – Defines Desired Behavior

QA – Writes Tests for Broken Behavior

Dev – Writes Tests for Working Behavior

QA finds broken behavior and proves that it’s broke.

Dev builds behavior and proves that it works.

Biz sets the requirements, priority, and approval standard of behaviors deployment.

Dev Automation is not out to get QA or somehow replace QA. Dev Automation takes the work product of QA and ensure that Dev is meeting Biz standards. Without QA Dev Automation would grossly increase cost.

Dev has a different goal than QA and they are supposed to be diametrically opposed in order to have the best check and balance. Two opposing viewpoints focused on one goal. It requires compromise, communication, and candor. It doesn’t require us to be at each other’s throat for the fear of looking bad. QA shouldn’t feel the heat when something is found to be broke. They should just document the break (write a test for it) and report the break to Dev. Prod Support tickets should go to both QA and Dev. Dev shouldn’t feel the heat when something is found to be broke. They should just document the fix (write a test for it) and submit the fixed behavior to QA. Dev should work off of QAs test because they are the pros at test.

The less QA finds broke the deeper they get to look and the more they can focus on issues such as security and performance.