TDD Testing Framework

For DemandWare

# Overview

If you’re like me you like to apply a test driven development approach to software development. The Demandware platform doesn’t have anything out of the box for testing your scripts or pipelines, but that doesn’t mean you can’t develop code in a TDD manner. That’s why I created this framework.

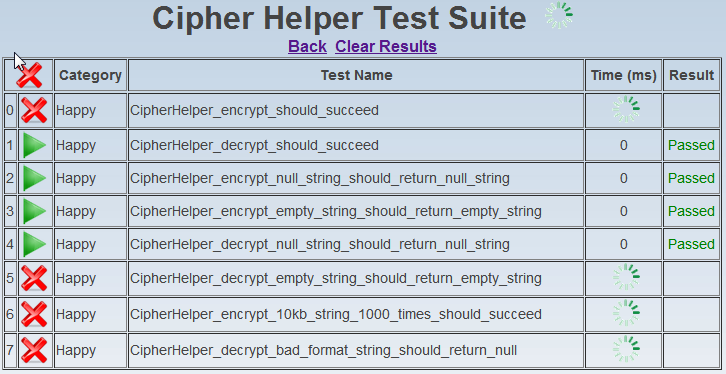
# What the framework looks like

The following images show:

1. The test suite listing page that shows all the test suites you currently have (*Figure 1*)
2. The tests running in the browser (*Figure 2*)
3. The results of running a test suite (*Figure 3*)



Figure



Figure

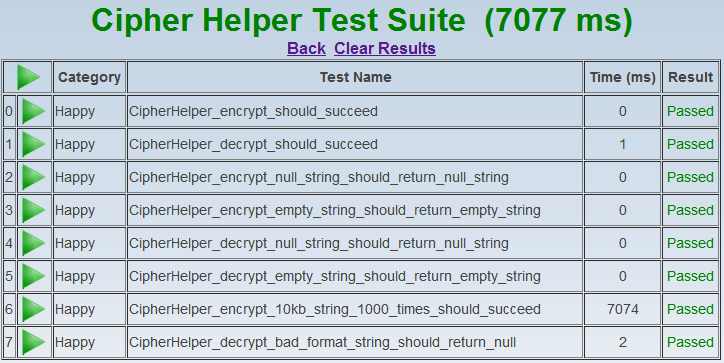


Figure 3

Clicking on the first green arrow  at the top will run all the tests in parallel (as much as the browser allows) and clicking on the subsequent arrows will run each test that the arrow is pointing to *(Figure 4 & 5).*

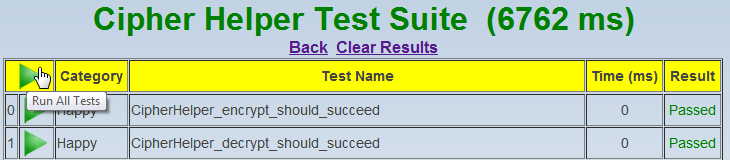


Figure 4

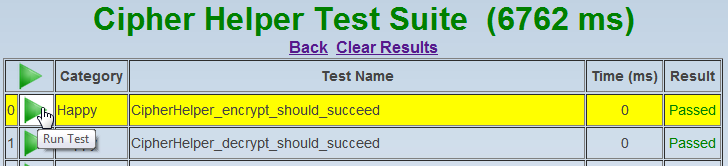


Figure 5

Clicking on the first red X will cancel all running tests and the subsequent X’s cancel the individual test that is running *(Figure 6)*.

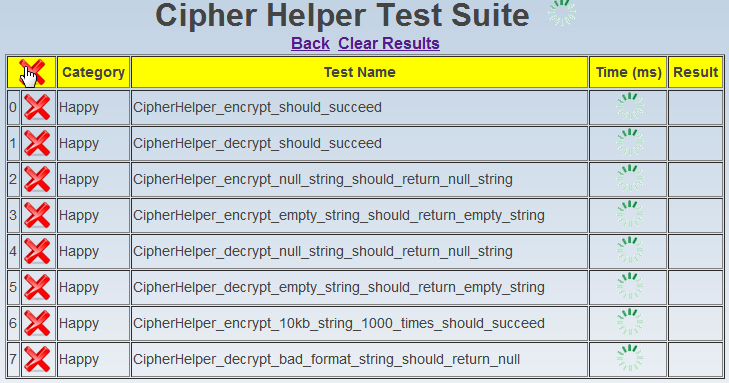


Figure 6

The following image shows a failing test (*Figure 7*)

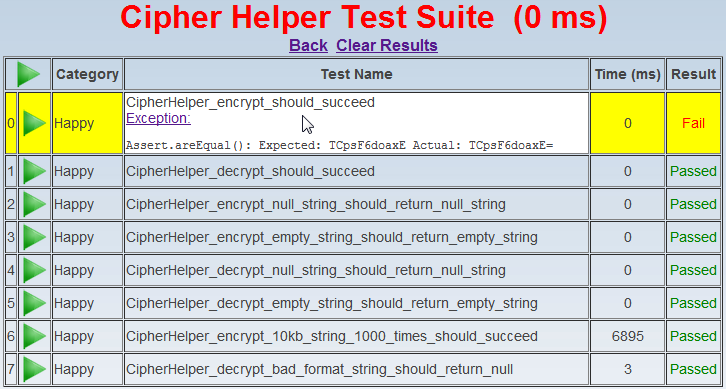


Figure 7

Notice how the title is in **RED** when a test fails – if all tests pass then the title will change to **GREEN**. Clicking on the ***Exception*** hyperlink will hide or show the reason why the test failed.

Any output from using the Logger.debug() statement in your test will also be shown *(Figure 8)*. Clicking on the ***Scriptlog*** hyperlink will hide or show the output from the Logger.debug() call.

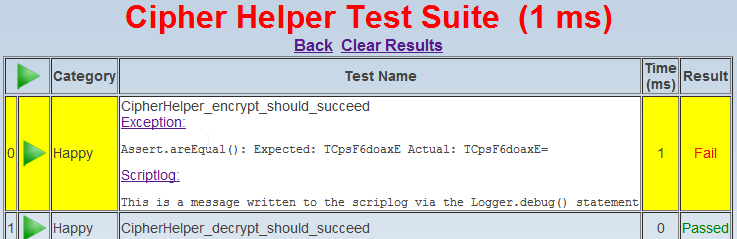
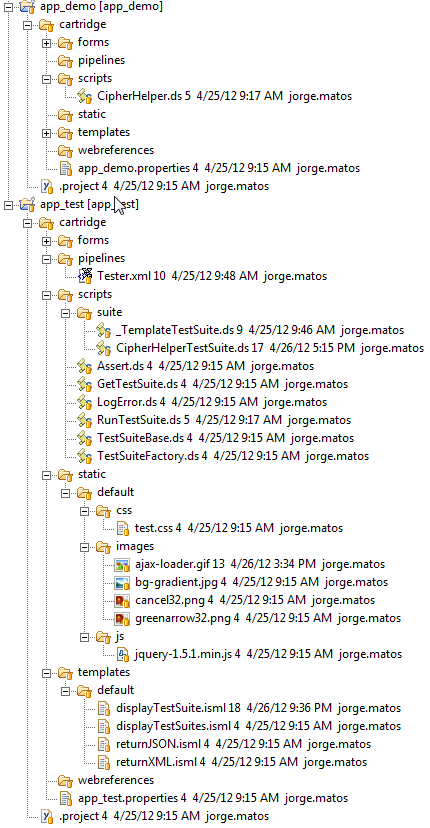


Figure 8

# What’s included in the Zip

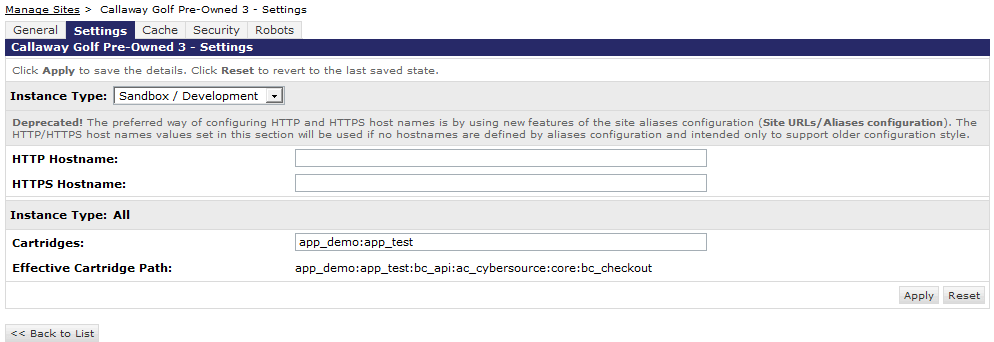
I’ve created a cartridge called ***app\_test*** that allows me to test Demandware script code and even pipelines. There’s another cartridge called ***app\_demo*** that has a single test script called ***CipherHelper.ds*** that I’m including as an example script that can be tested by the TDD framework.



All the code for the testing framework is in the ***app\_test*** cartridge.

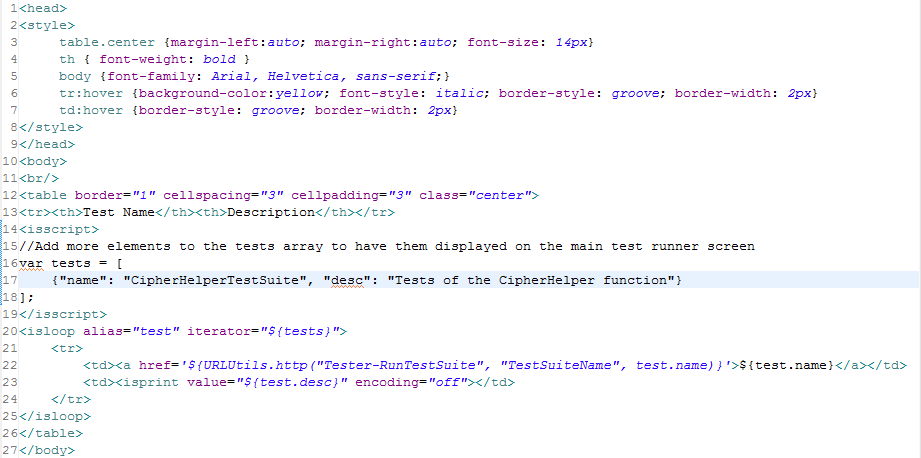
To start using the TDD framework you will need to do the following:

1. Add the ***app\_test*** cartridge to your workspace (and optionally ***app\_demo*** if you want to run the tests for the *CipherHelper.ds*)
2. Assign the ***app\_test*** (and optionally ***app\_demo***) cartridge to a website instance within the BusinessManager (see image below) under Administration > Sites > Manage Sites > Settings

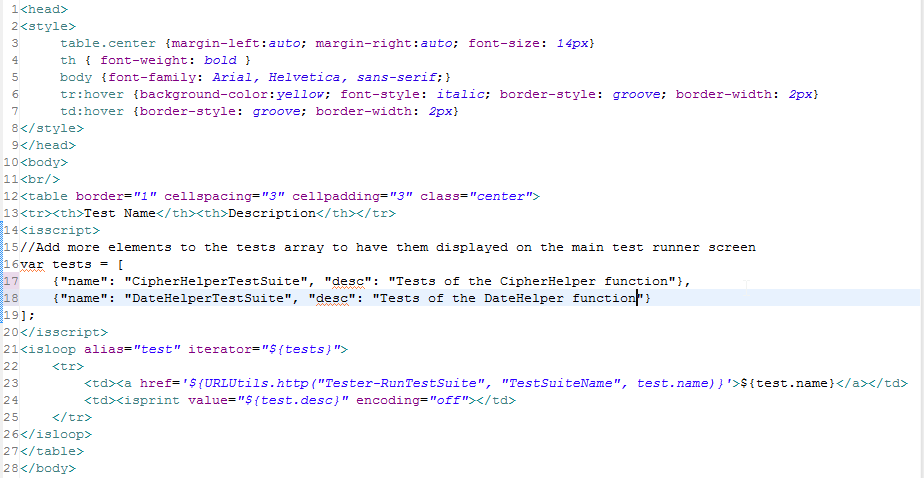


# Writing a TestSuite

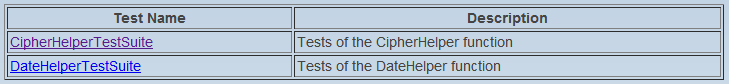
First you need to update the ***displayTestSuites.isml*** so that it displays your test suite on the main test page, the following is a snapshot of the ISML file:



You just need to add a new element to the ***tests*** array, for example if I wanted to add a new test suite called ***DateHelperTestSuite***:



The pipeline that you need to invoke to display the test suites is called ***Tester-Start***, if you invoke it in the browser you should get the following:



If you click on the ***DateHelperTestSuite*** link you will get the following:



Notice that the page does not display any tests for the ***DateHelperTestSuite.ds***, that’s because the file does not exist yet.

In the ***app\_test*** cartridge create the ***DateHelperTestSuite.ds*** in the ***app\_test:scripts/suite*** folder (this is where the framework expects to find all the test suites), the following is an example that shows the structure of the ***DateHelperTestSuite.ds*** script.



Notice the ***importScript(“app\_demo:DateHelper.ds”)*** at the top? That’s how the test suite gets access to the function being tested, in this case the ***DateHelper*** function.

The ***DateHelperTestSuite()*** is a function that uses a Power constructor function called ***TestSuiteBase*** and that function returns an object that you need to augment with the following members:

1. ***getName():***  function - is where you define the name of the test suite and is displayed on the web page.
2. ***tests*** : array - is an array of literal objects that represent each test in your test suite.

The testing framework iterates through all the objects in the ***tests*** array and invokes the ***run*** member to actually execute the test.

You can have as many tests as you want, just keep adding more literal objects to the ***tests*** array.



The ***Assert*** function is included in the framework and contains an assortment of methods to test the results of your tests (*Assert.isTrue*(), *Assert.areEqual()*, etc….) . The ***Assert*** function throws an exception that the framework catches - and that’s how it knows when a test has failed (the TDD framework catches any exception thrown by your test code and fails the test).

Each object that you create in the ***tests*** array can have the following members:

1. **name**: the name of your test, I like to use the “\_” character for spacing (required)
2. **category**: the category for the test, I use “Happy”, “Assert” and “Error” (required)
3. **ignore**: if this is true the framework will not run the test (optional)
4. **run**: a function that represents the code for your test (required)
5. **expectedError**: this is a string or regex expression that the framework will look for if an exception is thrown during the test – this is useful to test for cases where you expect an error to be thrown. (optional)

Here is another example of how to set up a literal test object:

{

"**name**": "FindConsumer\_with\_empty\_email\_should\_fail\_assertion",

"**category**": "Assert",

"**expectedError**": "Assert.isNotEmpty(): args.email was empty!",

"**ignore**": false,

"**run**": **function** () {

**var** request = {

"email": ""

};

WebServiceFactory.getService("sso").findConsumer(request);

Assert.fail("An assertion should have been thrown");

}

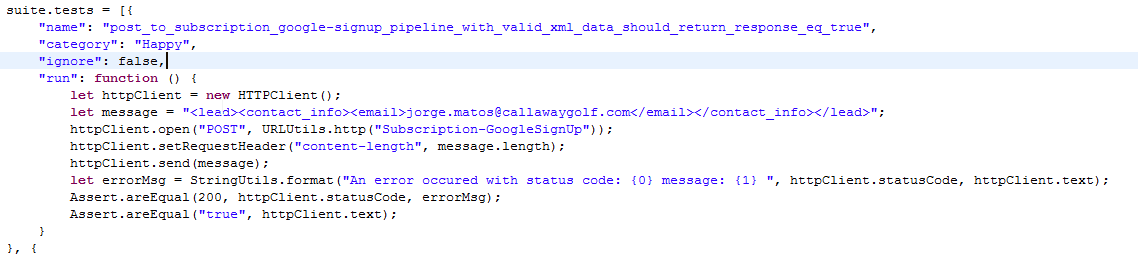
}

# Additional Info

## Testing Pipelines

In order to test a pipeline with this framework you can use the Demandware [***HTTPClient***](https://documentation.demandware.com/display/DOCAPI2115/dw.net.HTTPClient) API class to issue an HTTP GET or POST to your pipeline and you can use the ***Assert*** function to verify that the results were correct.

In the following example the test is posting to a pipeline and sending some XML data and the **statusCode** and **text** properties of the **HTTPClient** object are being checked for success (HTTP response 200 = OK).



## Running Tests from an Automated build

If you can issue an HTTP request from you automated build then you can run all (or some) of the tests and get back XML or JSON data.

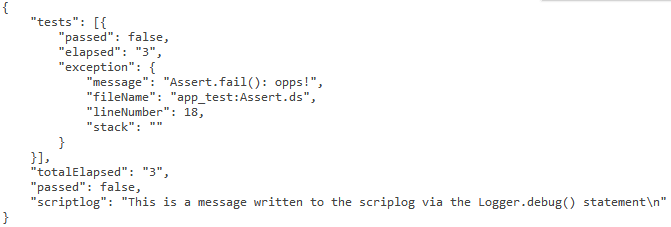
The following URL will return JSON data for the first test in the **CipherHelperTestSuite** test suite.

[http://[yourhostname]/on/demandware.store/Sites-[yoursite]-Site/default/Tester-Run?**suite**=CipherHelperTestSuite&**testid**=0](http://[yourhostname]/on/demandware.store/Sites-%5byoursite%5d-Site/default/Tester-Run?suite=CipherHelperTestSuite&testid=0)

Querystring parameters:

* **suite** – the name of the test suite being tested (required)
* **testid** – the ID number of the test as displayed in the user interface, use a value of **-1** to run all the tests in the suite (required)

JSON Output:



* **totalElapsed** – the total time in milliseconds to run all tests
* **passed** – true if all test have passed or false if one test failed
* **scriptlog** – the message passed to the Logger.debug() statement if used
* **tests** – an array of test results, each element contains:
  + **passed** – the individual test passed
  + **elapsed** – the time it took for the individual test
  + **exception** – if an exception occurs during the test (i.e. Assertion failure)

The following URL will return XML data for the first test in the **CipherHelperTestSuite** test suite.

[http://[yourhostname]/on/demandware.store/Sites-[yoursite]-Site/default/Tester-Run?**suite**=CipherHelperTestSuite&**testid**=0&**format**=xml](http://[yourhostname]/on/demandware.store/Sites-%5byoursite%5d-Site/default/Tester-Run?suite=CipherHelperTestSuite&testid=0&format=xml)

XML output:



Querystring parameters:

* **suite** – the name of the test suite being tested (required)
* **testid** – the ID number of the test as displayed in the user interface, use a value of **-1** to run all the tests in the suite (required)
* **format** – a value of “xml”, causes the output to be in XML format (optional)

# Dependencies & Disclaimer

The TDD framework makes use of the open-source [jQuery](http://jquery.com/) javascript framework, for manipulating the browser DOM and creating AJAX requests.

You are welcome to use the TDD framework as-is for whatever purpose you require. I’ve tested this framework pretty well and I use it quite often but if you find any bugs please let [me](mailto:Jorge.matos@callawaygolf.com?subject=Feedback:%20TDD%20Framework%20for%20Demandware) know and by the way, please don’t hold me or Demandware responsible if you shoot yourself in the foot using this code ☺.

*Peace,*

*Jorge Matos*