A Unit-Test Framework

For an Oracle Fusion Middleware environment

Intended Audience:

Developers

Author:

Jang-Vijay Singh

jv@singhpora.com

February 2018

Objectives

- When an incremental change is made (such as a bug fix) – repeatable tests add some assurance that existing functionality works as is
- Unit tests as developed currently mainly contain “inputs” and matching “expected outputs” (*but can test more complex scenarios too as shown later)*

The “expected outputs” were initially obtained from the versions of code prior to the change (e.g. from the 11g server prior to 12c migration or from the older version of code without the change).

- Without access to a business requirements catalogue, it is impossible to create a full set of test cases but developers can keep adding to these as information becomes available or as new features/bug fixes are developed.

Objectives (contd)

- It is too time-consuming to run unit tests such as these **manually** every time a change is made (and these only relate to **one** field)
- It's better to automate and include such tests

A	B	C
UCN		Expected
999990001	PlugReq001_NoTemperaturesSet.xml	Keep original value in PLUGREQ
999990002	PlugReq002_MinTempSet_PlugReqBlank.xml	Set PLUGREQ to Y as it was blank but MinimumTemperature (or Max Temp) is set
999990003	PlugReq003_MinTempSet_PlugReqSetN.xml	Keep original value in PLUGREQ
999990004	PlugReq004_MinTempSet_PlugReqSetX.xml	Keep original value in PLUGREQ (because it is explicitly set and not blank)

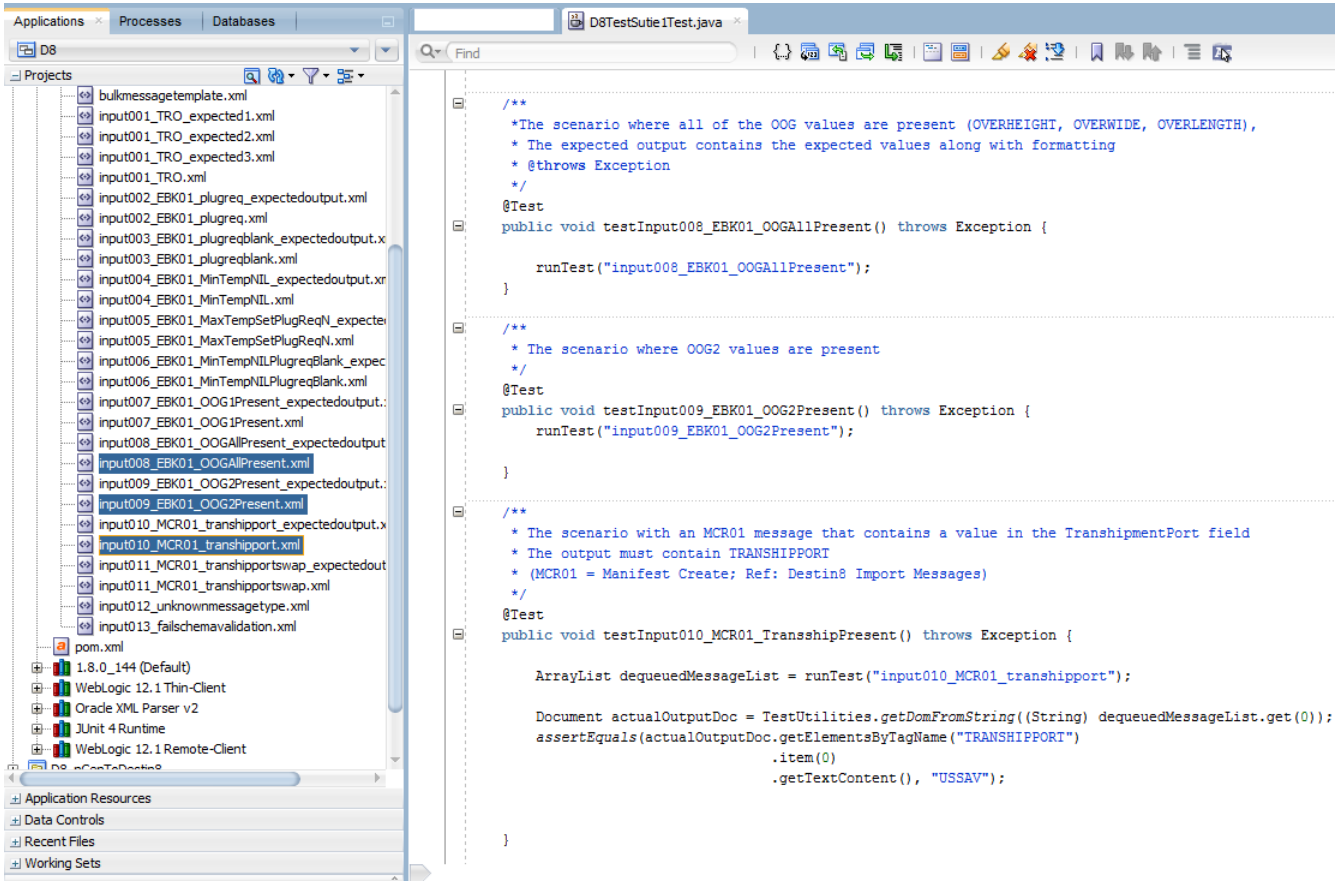
Principles

- Test a ‘unit’ – treating the system as a black box
- No point testing the platform features
- Test code is not shipped – it is only a local harness mainly used by developers to run tests repeatedly
- The goal is not to test network connectivity and environment configuration as such – it is to test code that is making changes to messages or applying some additional conditions/rules – this is why, it is better to mock external endpoints during **unit** testing
- It is **not a replacement** for systems integration testing (**SIT**) or user testing (UAT)
- As with every aspect of Software Engineering, one needs to apply pragmatism
- Test coverage can become a “***source of truth***” for *most* business requirements that were actually implemented *(this does* **not** *replace* ***formal business requirements specifications****)*
- *Migration of existing flows to upgraded technologies becomes more reliable (e.g. 11g -> 12c -> PaaS/iPaaS/Integration Cloud)*
- Of course, good test coverage leads to higher maintainability i.e. reliable change control and enhancements

Walkthrough Samples

An example test suite with test inputs and outputs – shown in JDeveloper

The simplest test case consists of just one line – calling a function “runTest” accompanied with input and output files – all the repeated code (send input to JMS, receive output from JMS, compare XML documents) is hidden in that function. More complex scenarios can be explicitly covered as well (testInput010 shown below)



Walkthrough Samples2

- Scenario: JMS to http to JMS

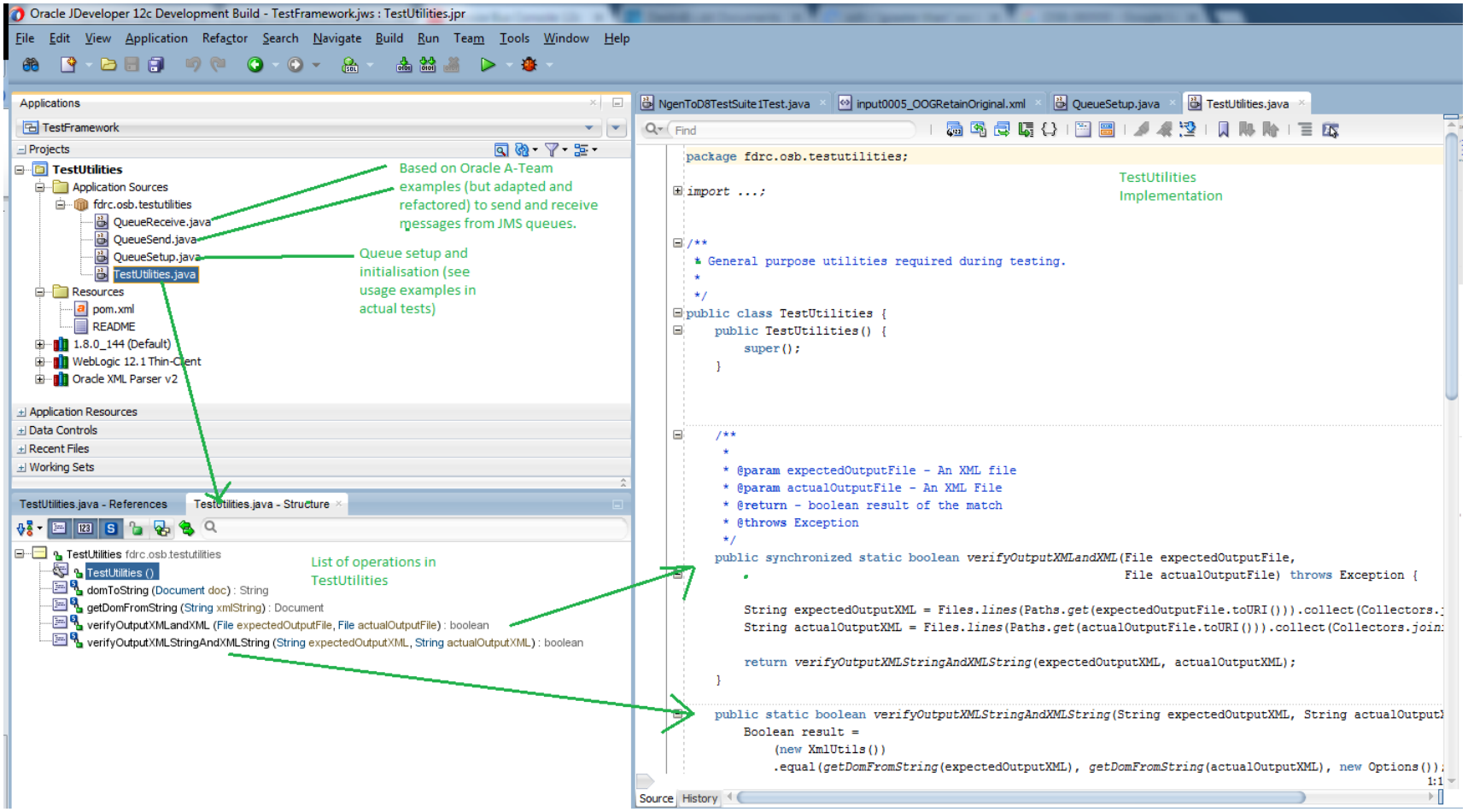
It helps to mock the external endpoint to keep tests repeatable and under controlled conditions.

WireMock is a good tool for mocking http endpoints - it can be started up with each test to supply specific outputs in different scenarios

Test Framework

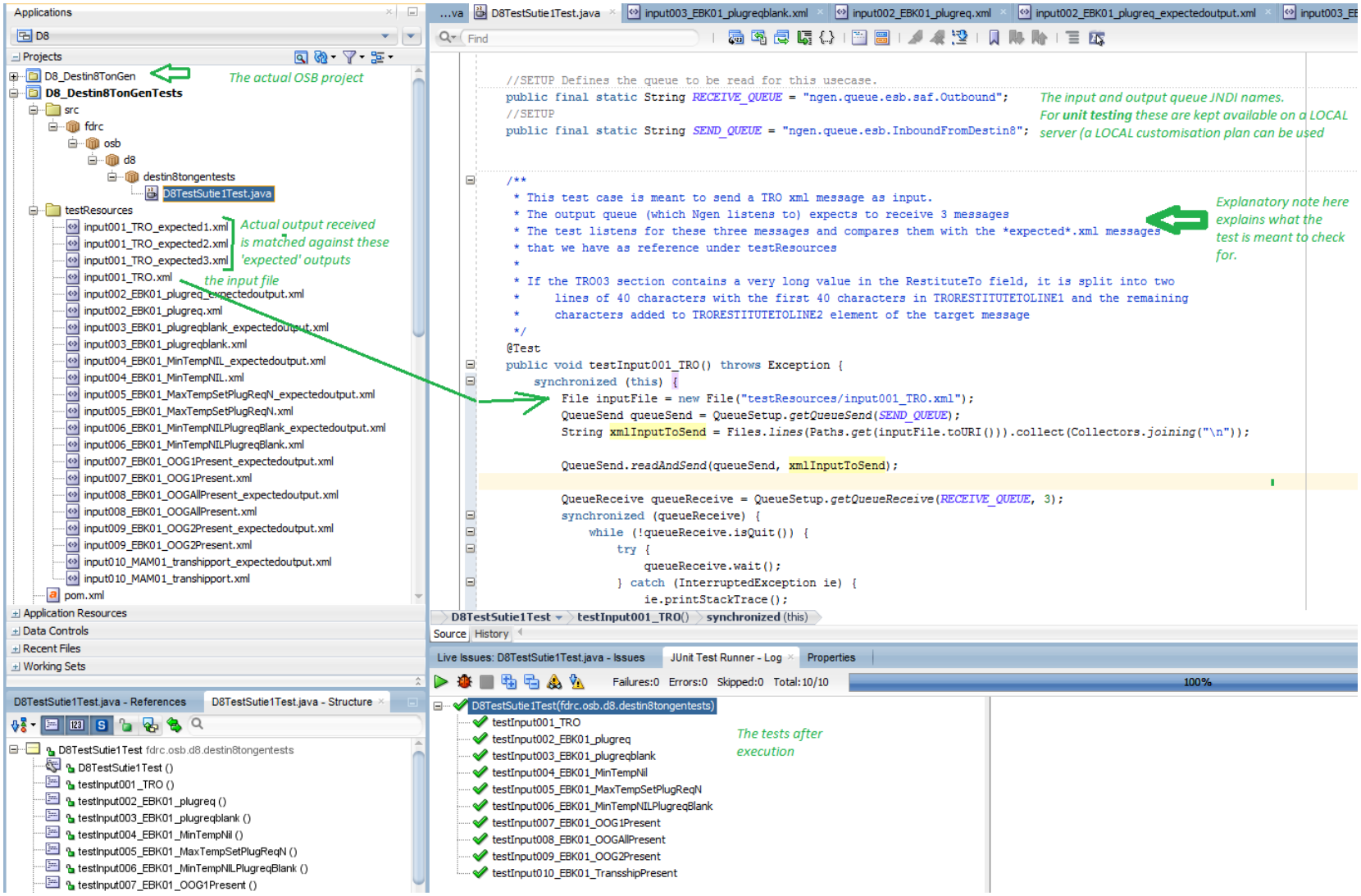
- A collection of reusable code that can be referred from individual projects.

Example:



Test Framework

- Example 2 – testing a publish-subscribe flow that includes JMS



Technologies

- Junit – plain and simple
- Oracle XMLDiff API's
- Oracle JMS samples from A-Team
- SOAPui – can be plugged in to automated tests