

Lightweight testing of heavyweight IBM Message Broker solutions

Summary

This article shows a hands-on approach of how you can test your IBM WebSphere Message Broker solutions in a simple way using modern Groovy and Java tools.

The approach taken could actually be used with more or less any integration platform although some actually do have built-in ways of doing it, like Apache Camel.

Introduction

Testing distributed enterprise integration using in our case IBM WebSphere MQ and IBM WebSphere Message Broker (IBM call it for their advanced ESB) is no so easily done as it ought to be.

The built in support for testing is not helping us much (see Test Client in resources).

At my current assignment there are a wealth of different approaches: JMeter, SoapUI, RFHUtil, MbTest, JUnit, end-to-end user testing, in-house developed testing tools.

I want a better approach that works well with our CI server and here is the start of that journey. To showcase I will use a vanilla IBM WebSphere Message Broker with default configuration (see appendix for links) and then one of the basic application samples for coordinated request/reply.

Imagine all you had to write was...

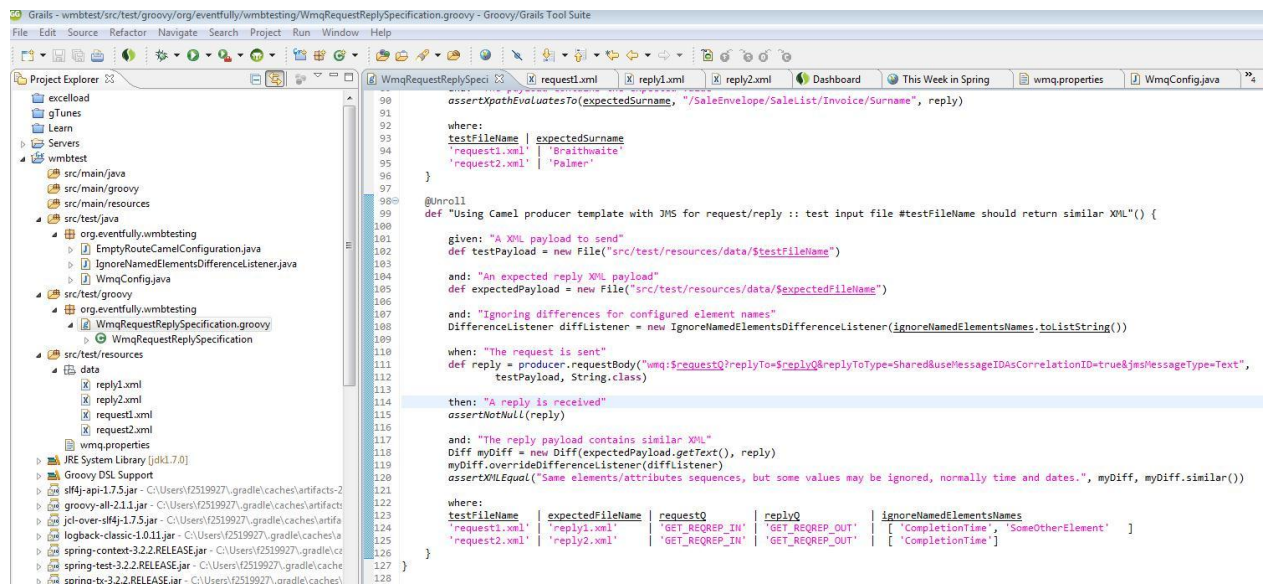
```
where:
testFileName      | expectedFileName | requestQ      | replyQ      |
ignoreNamedElementsNames
'request1.xml' | 'reply1.xml'      | 'GET_REQREP_IN' | 'GET_REQREP_OUT' | [
'CompletionTime', 'e2']
'request2.xml' | 'reply2.xml'      | 'GET_REQREP_IN' | 'GET_REQREP_OUT' | [ 'CompletionTime'
]
```

To “inject” the data into the test that will be run once per row above (omitting the details)

```
given: "A XML payload to send"
and: "An expected reply XML payload"
and: "Ignoring differences for configured element names"
when: "The request is sent"
then: "A reply is received"
and: "The reply payload contains similar XML"
```

The actual code looks like this:

Figure 1 - The code in the IDE



To configure it with your queuemanager, just edit a simple properties file (src/test/resources/wmq.properties)

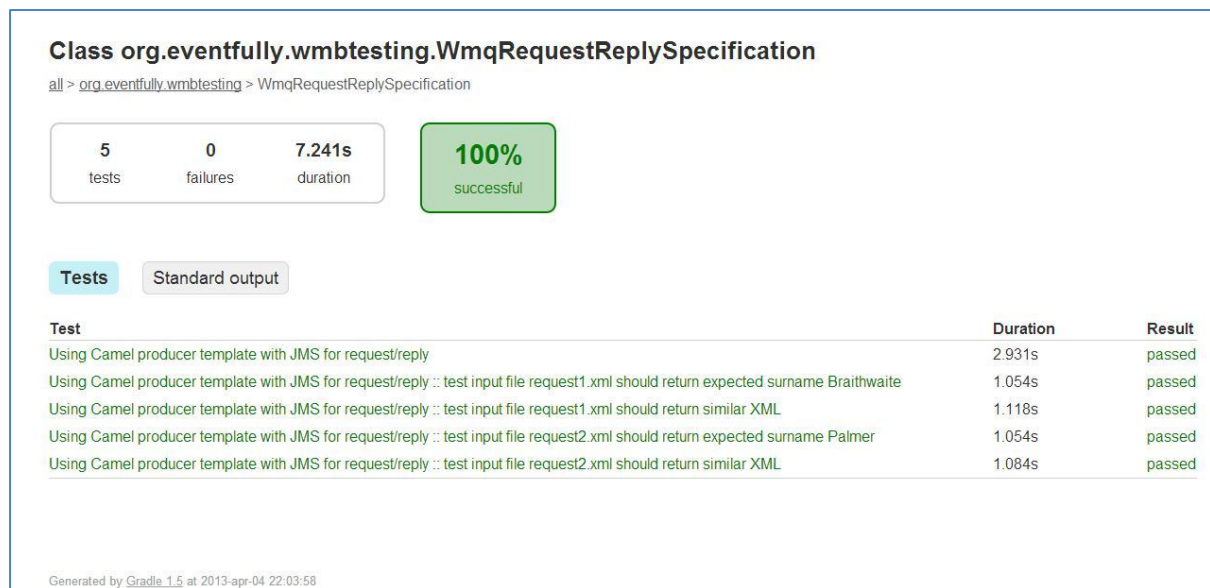
```
qmgr.hostName=localhost
qmgr.port=2414
qmgr.queueManager=MB8QMGR
qmgr.channel=SYSTEM.AUTO.SVRCONN
#0 = binding, 1 = client, 8 = first binding, then client
qmgr.transportType=1
```

Then to test it:

```
# Use gradlew.bat or gradlew if you don't want to install Gradle
gradle test
```

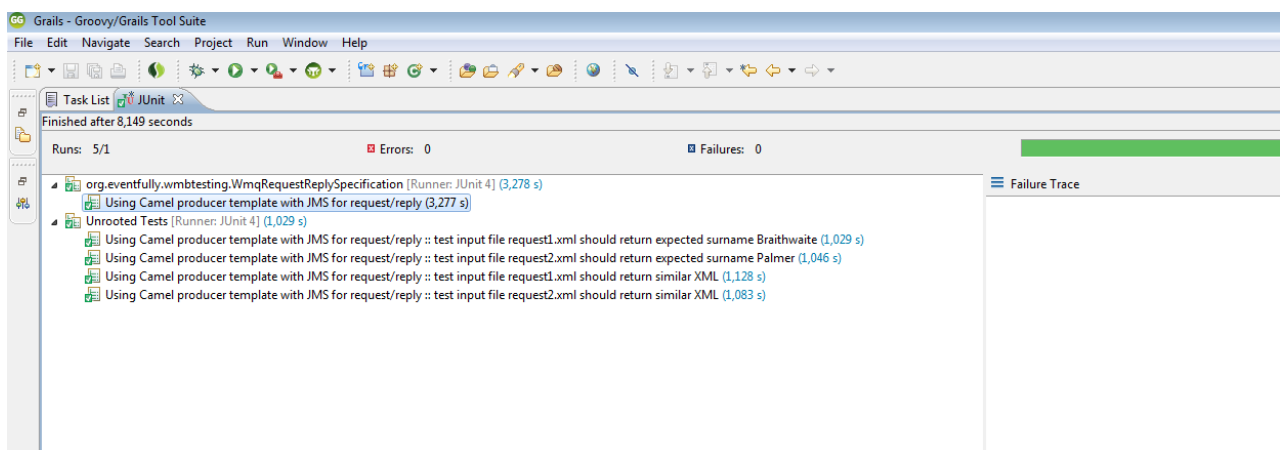
Check the results graphically in "build/reports/tests/index.html"

Figure 2 - Spock test results



If you prefer, you can actually run it as a JUnit test directly from your IDE instead.

Figure 3 - JUnit result from "run as JUnit test" in GGTS



Appendix A - Setting up the WMB samples

Summary with pictures:

Figure 4 - Start the WMBT with a new workspace and then start from samples

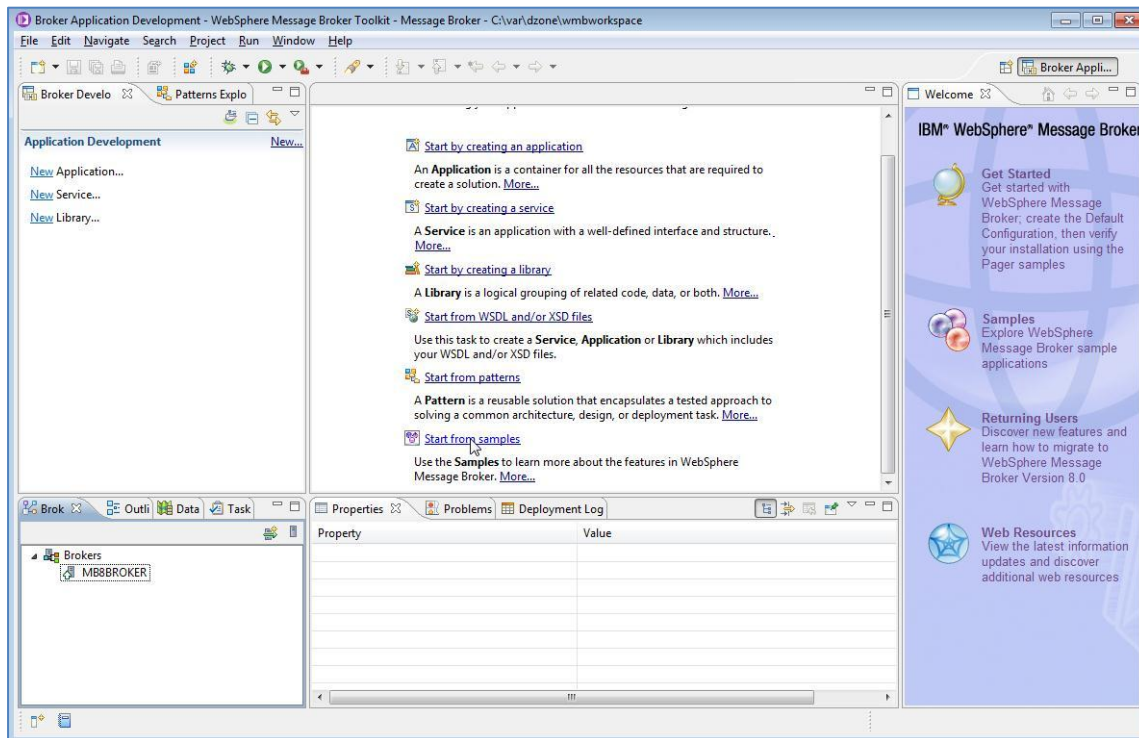


Figure 5 - Choose the "Coordinated Request Reply sample"

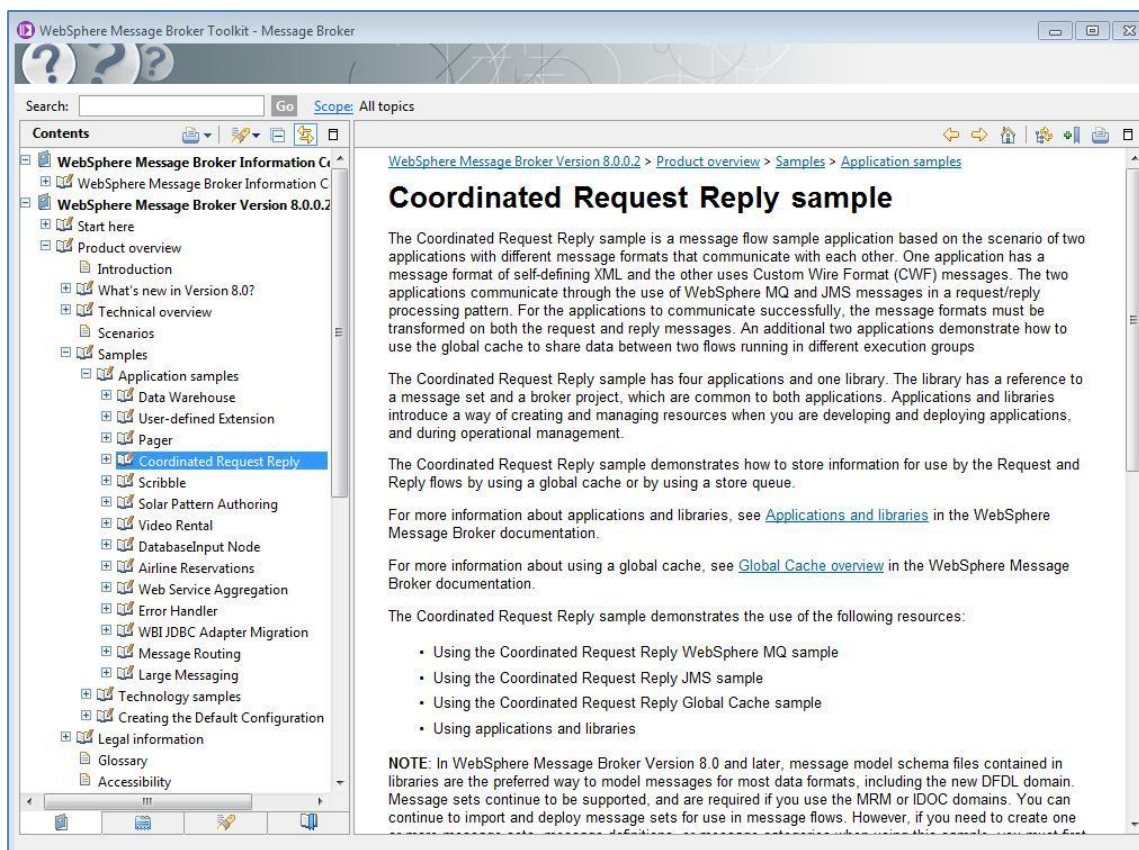






Figure 6 - Import and deploy the sample

To find out more about the sample and how to get the prebuilt sample running by using the wizards, click the following links.

 **Import and deploy:** 5 minutes

 [Read about the sample](#)

 [Setup instructions](#)

 You can set up the sample in one of the following ways:

- [Import and deploy the sample](#)

This option imports the sample files into your workspace and deploys the sample to the broker. This option also sets up additional resources for the sample, for example, WebSphere MQ queues. By default, the Coordinated Request Reply MQ application, Coordinated Request Reply Global Cache application and Coordinated Request Reply Backend application are deployed. The Coordinated Request Reply JMS application is not deployed because additional manual steps are required to configure the Coordinated Request Reply JMS application for testing.

Figure 7 - Import summary that all went well

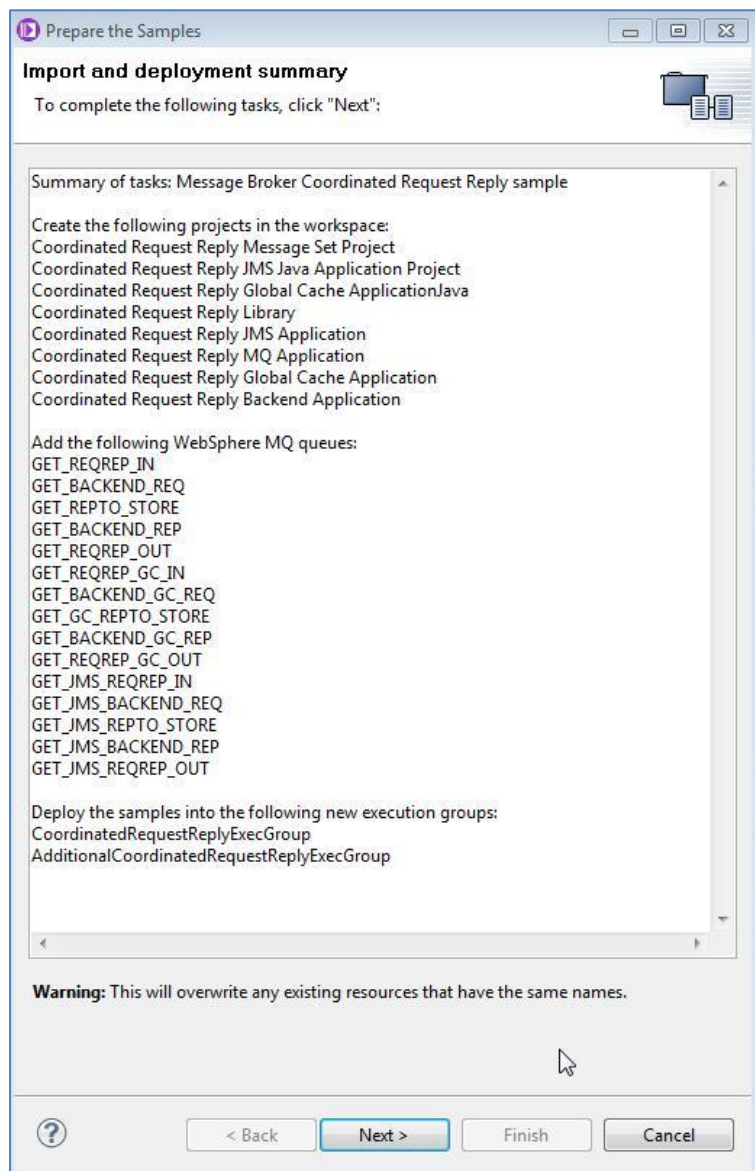
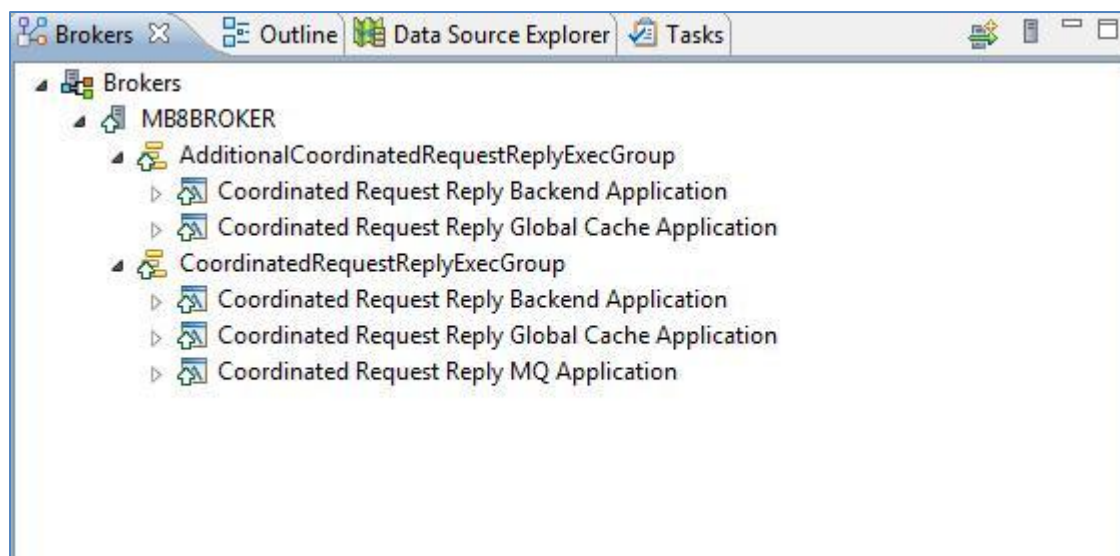


Figure 8 - Check that the applications are deployed to the Execution Groups



Appendix B - Environment used

In case we run into the infamous “works on my machine” syndrome.

IDE

Groovy/Grails Tool Suite – version: 3.2.0.RELEASE

Build tool – Gradle

Gradle 1.4

Gradle build time: den 28 januari 2013 kl 3:42 UTC
Groovy: 1.8.6
Ant: Apache Ant(TM) version 1.8.4 compiled on May 22 2012
Ivy: 2.2.0
JVM: 1.7.0 (Oracle Corporation 21.0-b17)
OS: Windows 7 6.1 amd64

IBM WebSphere MQ

Name: WebSphere MQ
Version: 7.0.1.8
CMVC level: p701-108-120224

IBM WebSphere Message Broker

BIP8927I: Broker Name 'MB8BROKER'
Queue Manager = 'MB8QMGR'
Operation mode = 'advanced'
Fixpack capability level = 'all' (effective level '8.0.0.2')

Operating System

OS Name:	Microsoft Windows 7 Enterprise
OS Version:	6.1.7601 Service Pack 1 Build 7601
System Model:	HP EliteBook 8540w
System Type:	x64-based PC

Appendix C - Additional resources and links

The source code for this article at GitHub

Download, clone or fork from:

<https://github.com/magnuspalmer/wmbtesting>

Spock framework

Spock is a testing and specification framework for Java and Groovy applications. Since it is JUnit under the hood, it works fine with many IDEs, build tools and CI servers.

<https://code.google.com/p/spock/>

Gradle enterprise build automation

The build, test and deployment killer app, bye bye Maven....

<http://www.gradle.org/>

Apache Camel

Lightweight integration framework powered by really nice DSL.

<http://camel.apache.org/>

XMLUnit

Java JUnit extensions that I think is great for comparing two XML documents.

<http://xmlunit.sourceforge.net/>

IBM WMB Test Client

The documentation at Info Center for V8 for the Test Client.

http://publib.boulder.ibm.com/infocenter/wmbhelp/v8r0m0/topic/com.ibm.etools.mft.doc/af52104_.htm

IBM WMB Setup default configuration

The documentation at Info Center for V8 for the default configuration.

http://publib.boulder.ibm.com/infocenter/wmbhelp/v8r0m0/topic/com.ibm.etools.mft.doc/ae20200_.htm