



LAB- TESTING

In this lab you will understand

- a) How to write JUNIT functional test case
- b) How to write unit and functional test cases using MUNIT

Step 1 - JUNIT FUNCTIONAL TEST CASE

In this step, you will be working on **01-junit-funtionaltestcase-start** project under **07-testing**

In this step, you will understand how to write functional test case using JUNIT.

1) Open testFlow.xml and observe the flow. It expects a query parameter username and returns 'hello'+username

We want to write a functional test case for this flow.

2) under src/test/java , create a java class Way2LearnTests in package "com.way2learnonline.tests"

Make this class to extend FunctionalTest case belonging to package org.mule.tck.junit4

Override a method getConfigFiles() as shown below :

```
@Override
protected String[] getConfigFiles() {
    return new String[]{"src/main/app/testflow.xml"};
}
```



Now Write a method with any name (e.g testFlow()) and annotate with @Test as shown below :

```
@Test
public void testFlow() throws Exception{
}
```

Write the following code in the method

```
MuleClient muleClient= muleContext.getClient();
DefaultMuleMessage message= new DefaultMuleMessage(null, muleContext);

MuleMessage responseMessage=
    muleClient.send("http://localhost:8081/test?username=siva",
message);

Object responsePayload=responseMessage.getPayloadAsString();
Assert.assertEquals("hellosiva", responsePayload);
```

Understand the above code and run the test case and make sure that it will pass.

Step 2 - MUNIT TEST CASES

In this step, you will be working on **02-munit-start** under **07-testing**

1) Open munitdemo.xml and understand the flow.

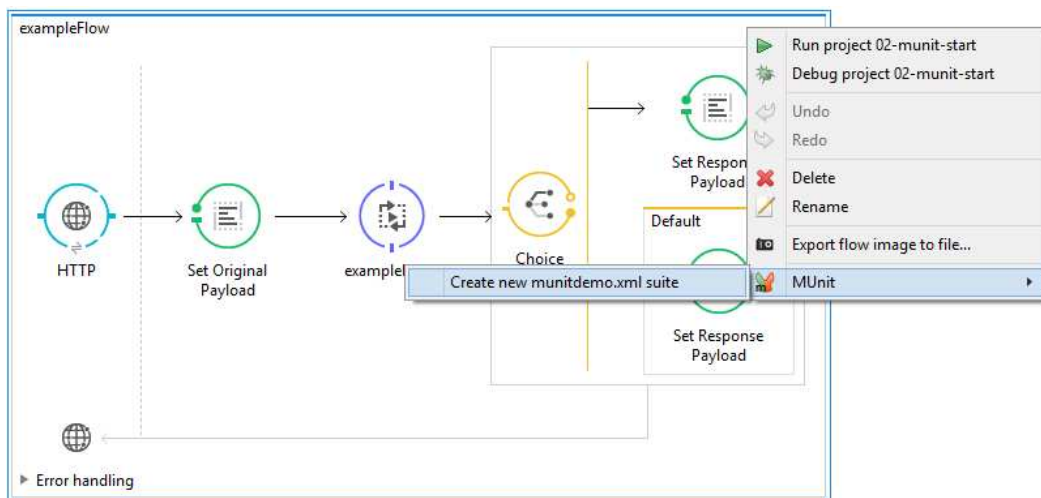
It expects a query parameter url_key and sets its value as payload.

At a high level, if url_key=payload_1, the flow returns response_payload_1 else it returns response_payload_2

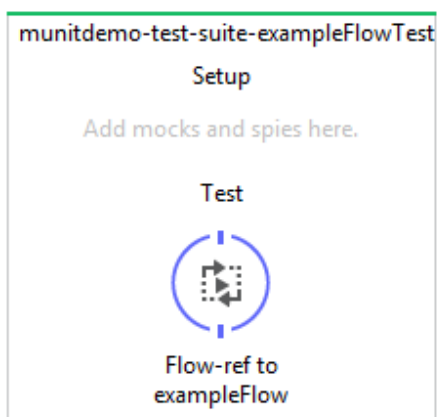


2) Now we want to write functional test cases for this flow.

3) Right click on the flow (on top of the flow just below the blue line not in middle of the flow) and select MUNIT-->create new munitdemo.xml suite as shown below :



A new file with name munitdemo-test-suite.xml will be created under src/test/munit . it looks as shown below :



Open the xml view. It should look like below :

```
<munit:config name="munit" doc:name="MUnit configuration"/>
  <spring:beans>
    <spring:import resource="classpath:munitdemo.xml"/>
  </spring:beans>
<munit:test name="munitdemo-test-suite-exampleFlowTest" description="Test">
```



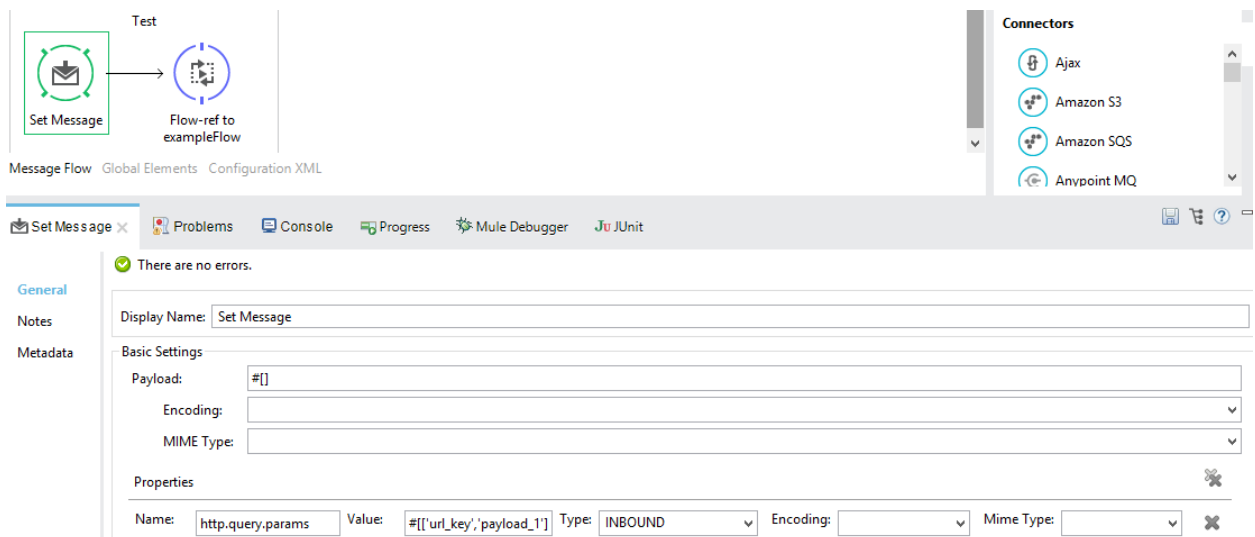
```
<flow-ref name="exampleFlow" doc:name="Flow-ref to exampleFlow"/>
</munit:test>
```

4)

Scenario -- VerifyCall

when a message comes to example flow, with inbound query parameter url_key=payload_1, we want to **verify call to "exampleFlow2"**

Drag a set Message component before Flow ref and set the payload as #[] . Click on Add Property button and configure property with name http.query.params and value as #[['url_key','payload_1']] and type as "inbound" as shown below :



5) Drag Verify Call after Flow Reference. Click on



and double click

exampleFlow2:Flow as shown below

Display Name:

Matching Processor Condition

Verify call of message processor that matches:

And attributes satisfy:

Name: Value:

☐ Message processor Verify Times

type filter text

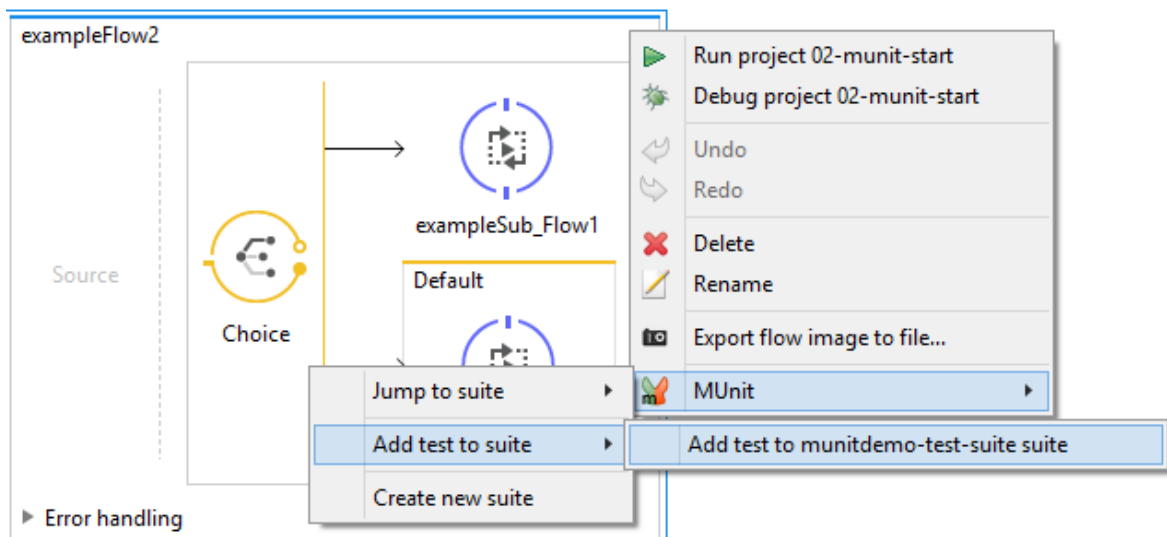
- ▾ : Mule Configuration
 - exampleFlow : Flow
 - exampleFlow2 : Flow
 - exampleSub_Flow1 : Sub Flow
 - exampleSub_Flow2 : Sub Flow

6) Run the Munit Suite and make sure it will pass.

Running MUNIT tests requires administrator privileges. So, make sure that you run Anypoint studio as Administrator. Otherwise you may get exceptions.

Creating another test case in same munit test file


7) Right click on exampleFlow2 (again on top of blue line) and select MUNIT-->Add test to suite --> Add test to munitdemo-test-suite suite as show below



8) A new Flow will be created in munitdemo-test-suite.xml

Drag "Set message" component before flow reference and set the payload as #['payload_1']




Drag Verify Call after Flow Reference. Click on  and double click exampleSub_Flow1:Flow as shown below


✓ There are no errors.

Display Name:

Matching Processor Condition

Verify call of message processor that matches: 

And attributes satisfy:

Name: Value: 

type filter text

- 🔗 : Mule Configuration
 - ▶ ☒ exampleFlow : Flow
 - ▶ ☒ exampleFlow2 : Flow
 - ▶ ☒ exampleSub_Flow1 : Sub Flow
 - ▶ ☒ exampleSub_Flow2 : Sub Flow

Run the Suite and observe that this second test fails.

Remember that for Subflows, you have to use `munit matcher matchContains` as shown below:

Also observe that we are using "name" instead of "doc:name"

Display Name:

Matching Processor Condition

Verify call of message processor that matches:

And attributes satisfy:

Name: Value:

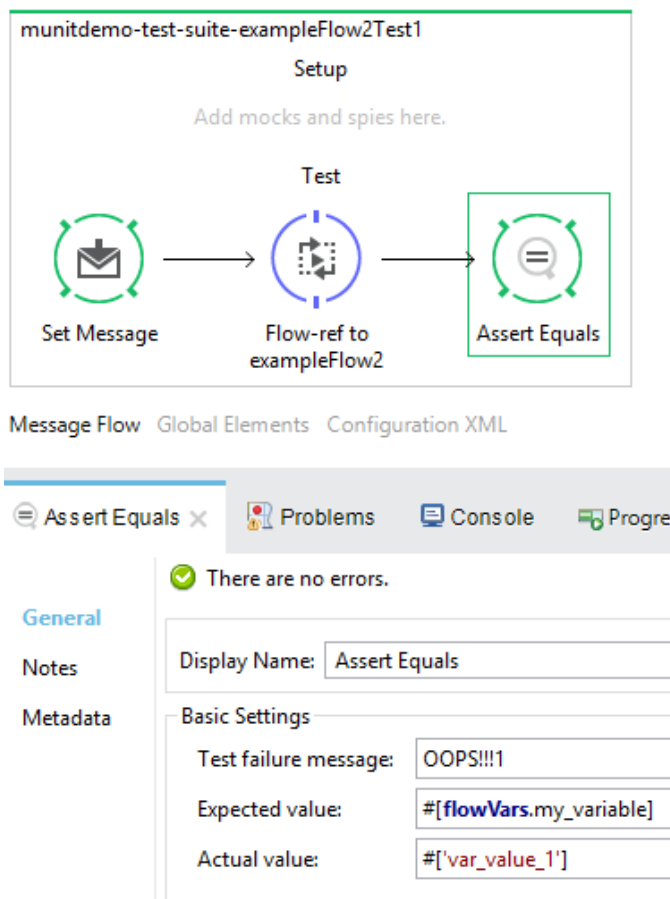


Creating another test case in same munit test file to verify the flow variable

Right click on exampleFlow2 (again on top of blue line) and select MUNIT-->Add test to suite --> Add test to munitdemo-test-suite suite as show below

Drag a set message component before flow reference and set the payload as `#['payload_1']`

Drag a AssertEquals component and configure it as shown below :



Run the Suite and make sure it will pass



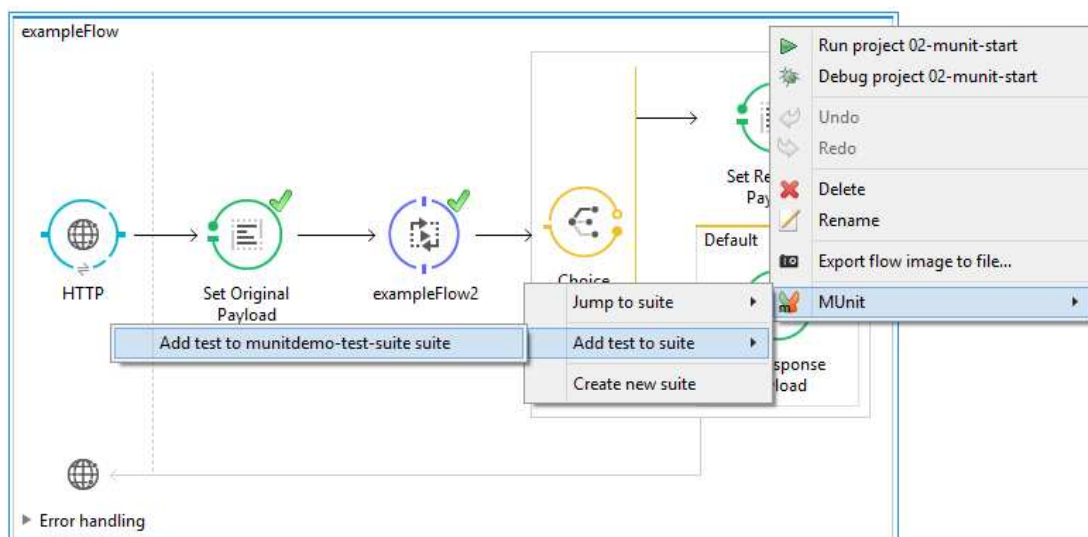
Writing Unit Test Cases by Mocking components

We want to write unit test for exampleFlow.

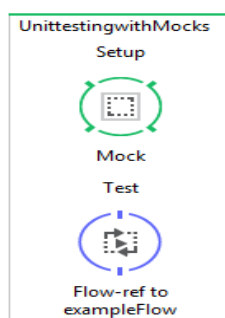
As this flow is having "set Payload Transformer" which is expecting query parameter, we have to mock it.

Also, the flow reference with name "exampleFlow2 " also has to be mocked.

Right click on exampleFlow (again on top of blue line) and select MUNIT-->Add test to suite --> Add test to munitdemo-test-suite suite as show below




Drag and drop a Mock component in the Set up portion of the flow as shown below:

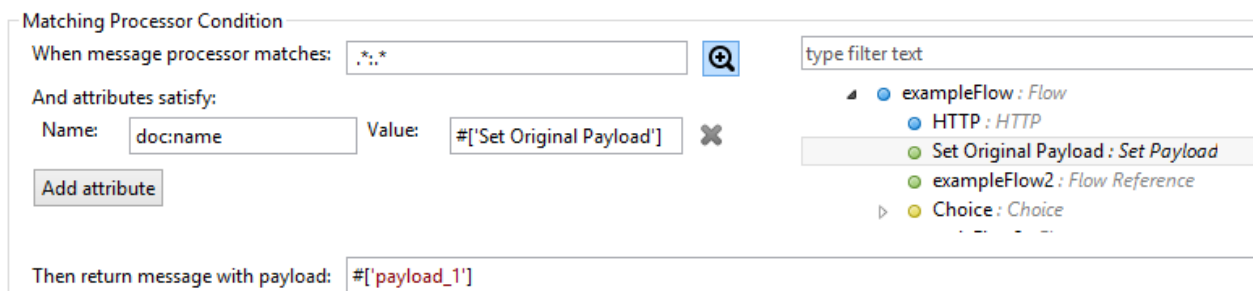





Double Click on the mock .

Click on  and double click on exampleFlow\set Original Payload .


Configure as return payload as #[payload_1] as shown below :



Matching Processor Condition

When message processor matches: 

And attributes satisfy:

Name: Value: 

Then return message with payload:

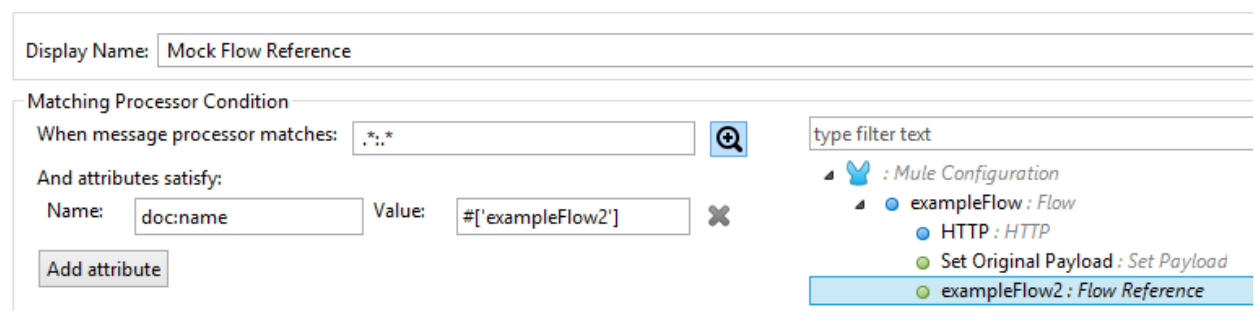
type filter text

- exampleFlow : Flow
 - HTTP : HTTP
 - Set Original Payload : Set Payload
 - exampleFlow2 : Flow Reference
 - Choice : Choice

Drag and drop another Mock component in the Set up portion of the flow


Double Click on the mock .

Click on  and double click on exampleFlow\exampleFlow2 as shown below :




Display Name:

Matching Processor Condition

When message processor matches: 

And attributes satisfy:

Name: Value: 

type filter text

- Mule Configuration
 - exampleFlow : Flow
 - HTTP : HTTP
 - Set Original Payload : Set Payload
 - exampleFlow2 : Flow Reference
 - Choice : Choice

Click on Add Property button at the bottom and configure it as shown below :



Matching Processor Condition

When message processor matches:

And attributes satisfy:

Name: Value:

Then return message with payload:

Encoding:

MIME Type:

Properties

Name: Value: Type: Encoding: Mime Type:

type filter text

- ⚡ : Mule Configuration
 - ⚡ exampleFlow : Flow
 - ⚡ HTTP : HTTP
 - ⚡ Set Original Payload : Set Payload
 - ⚡ exampleFlow2 : Flow Reference
 - ⚡ Choice : Choice
 - ⚡ exampleFlow2 : Flow
 - ⚡ exampleSub_Flow1 : Sub Flow
 - ⚡ exampleSub_Flow2 : Sub Flow

Now Drag Assert payload component after flow reference and assert the payload as `#['response_payload_1']`

Run the tests and make sure that it will pass