



# TGH

Making Integrations Simpler



# MUnit in MuleSoft

**Author**

**Yuvraj Sinha**



## Contents

1. Introduction .....	1
2. Assert expression, Set event & Mock when .....	4
3. Writing functional test case .....	14
4. Verify call .....	18
5. Before/After Suite & Before/After Test .....	22
6. Spy Event Processor .....	28

## Introduction

### What is MUnit?

MUnit is a Mule application testing framework that allows you to easily build automated tests for your integrations and APIs.

### Why use MUnit?

It provides a full suite of integration and unit test capabilities and is fully integrated with Maven and Surefire for integration with your continuous deployment environment.

With MUnit you can:

- Create your test by writing Mule code
- Mock processors
- Spy any processor
- Verify processor calls
- Enable or ignore particular tests
- Tag tests
- Check visual coverage in Studio
- Generate coverage reports

MUnit is fully integrated with Anypoint Studio, allowing you to create, design and run MUnit tests just like you would Mule applications.

### What is Unit testing?

Unit testing is a software testing technique in which individual units or components of a software application are tested in isolation from the rest of the system. A unit is the smallest testable part of any software, typically a function, method, or procedure. The purpose of unit testing is to validate that each unit of the software performs as designed and expected.

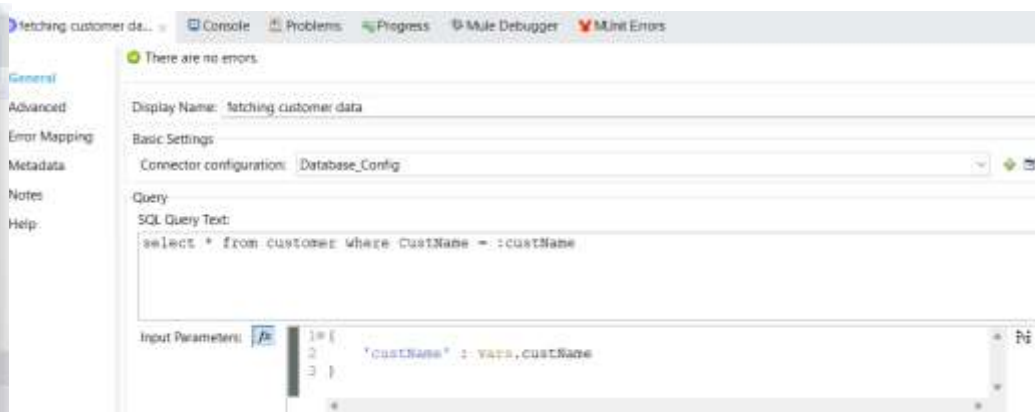
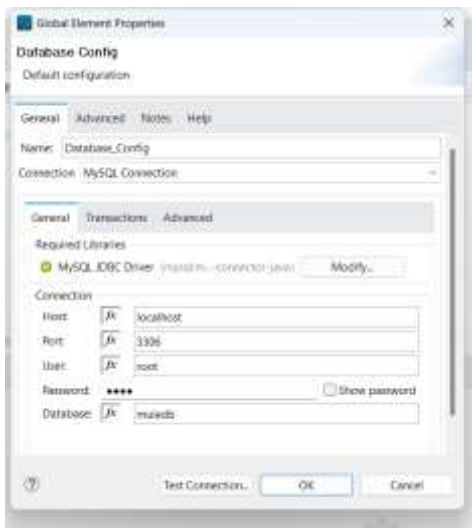
In unit testing, the developer writes test cases that cover different scenarios and edge cases for each unit of code. These test cases are automated and run frequently during the development process. The goal is to ensure that each unit behaves correctly and produces the expected output for various inputs.

©[TGH Software Solutions Pvt. Ltd.](#)

## The application we are going to test is



## Database configuration



## postman



©TGH Software Solutions Pvt. Ltd.

No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent



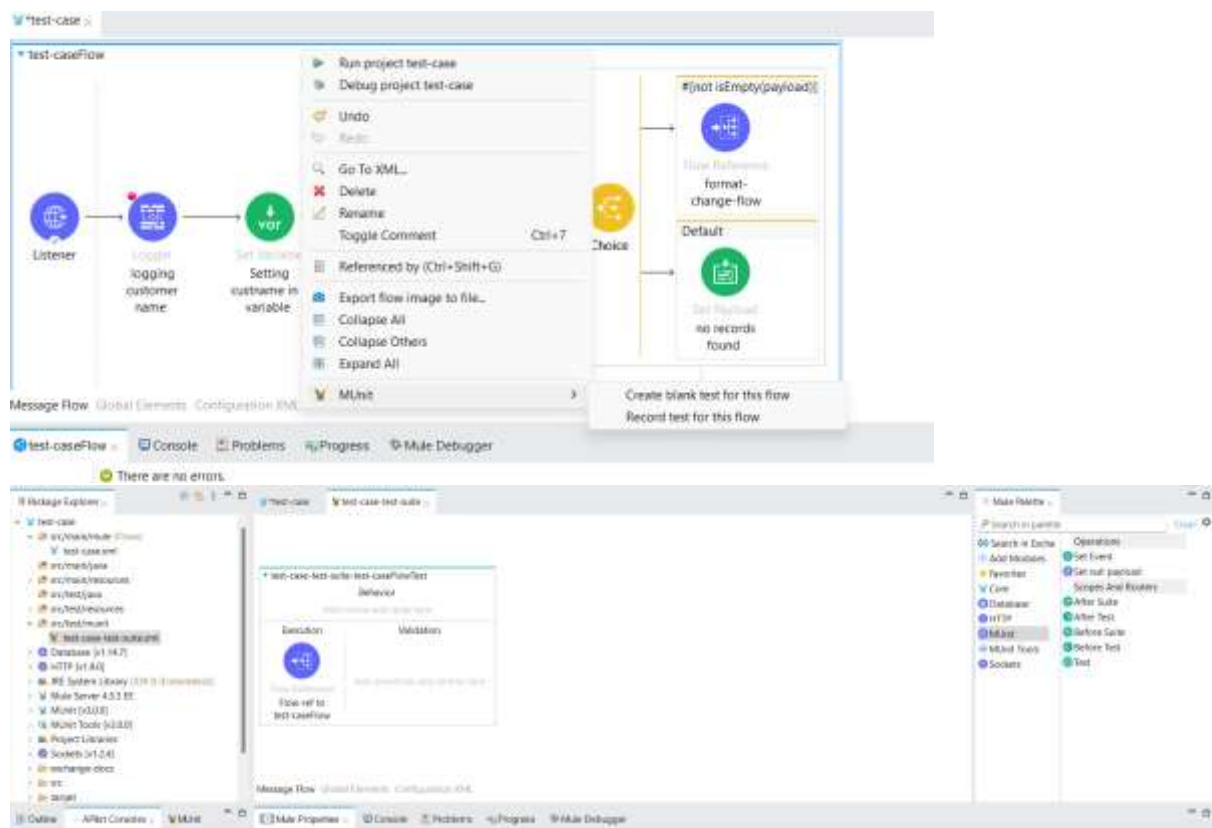


## Assert expression, Set event & Mock when

### Objective:

We are going to write a Unit Test case for the above scenario (postman request and response)

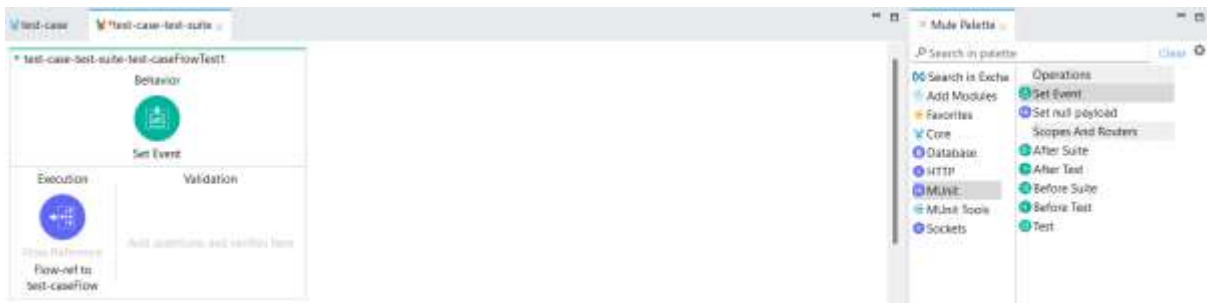
**Step -1:** Click on the main flow that you want to write test cases (test-caseFlow) and then right-click on it. Select “MUnit” and then click “Create blank test for this flow”. This creates a blank test flow where you can write your test cases and also it creates a test-case-suite.xml.



**Step 2:** Drag and drop a set event processor from the mule pallet in the MUnit module onto the behaviour section of the test flow.

The Set Event Processor allows us to define a Mule Event.

This message processor is normally used at the beginning of an MUnit test, to define the first message to send to the flow being tested.



**Step 3:** Now we are going to create an event with an empty payload and attributes should contain 2 query parameters “custName” with the value Yuvraj and format with the value “json”.

Configure

```
{'queryParams': {'custName': 'Yuvraj', 'format': 'json'}}
```

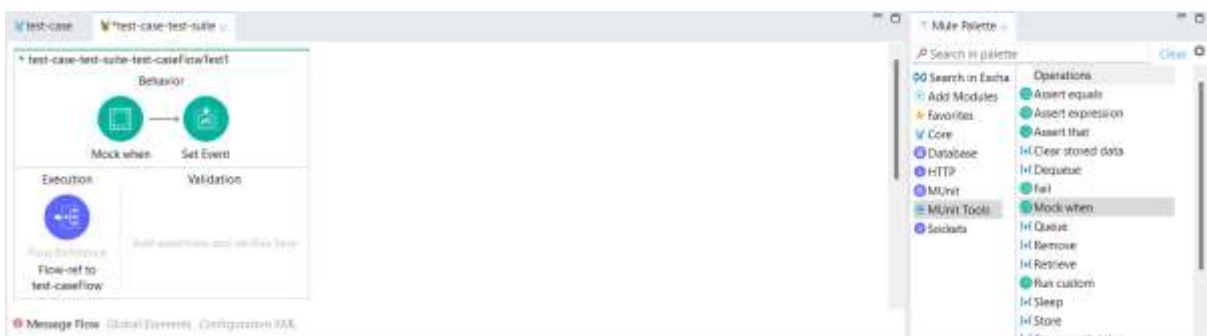
as the value for attributes in the set event.

(You can set the event according to the request you’re calling it with)

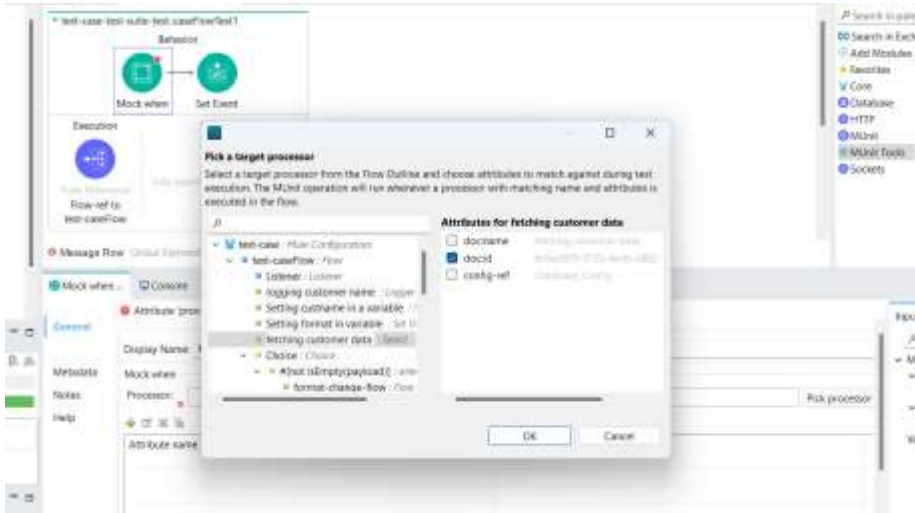


**Step 4:** Now we are going to mock call the “fetching customer data”

Drag and drop the mock when on the behaviour section of the flow



Now click on the “pick processor” and select the component you want to mock (fetching customer data) select its “doc:id” and then click “ok”



**Step 5:** Now set the value of payload in “Then return”, with

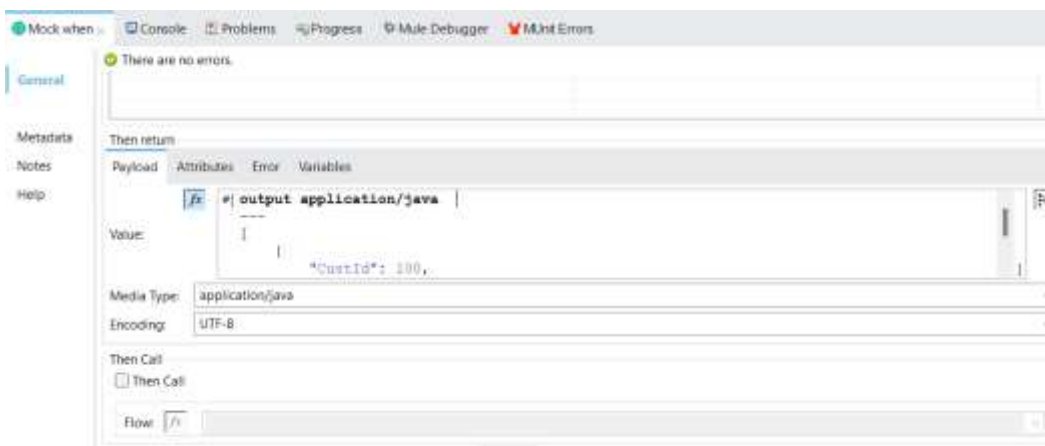
**output application/java**

```

---
[
  {
    "CustId": 100,
    "OrderQty": "10",
    "CustName": "Yuvraj"
  }
]

```

And the media type to application/java



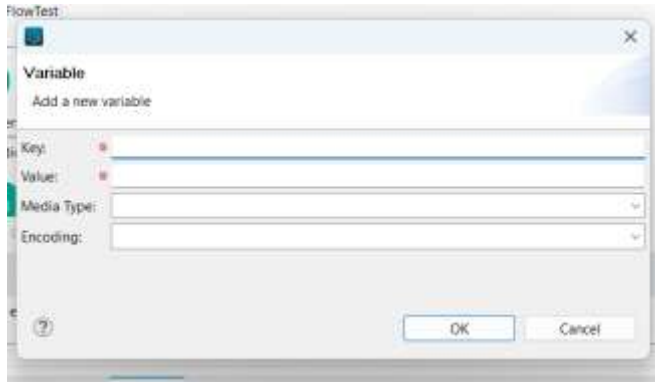
Now go to the variables section and click on the ‘+’ button. It will appear as below

©[TGH Software Solutions Pvt. Ltd.](http://TGH Software Solutions Pvt. Ltd.)

No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent







Now fill up the section with values below and click “OK”

Key = custName

Value = #["Yuvraj"]

Media Type = application/java (select from the drop-down)

Encoding = UTF-8 (select from the drop-down)

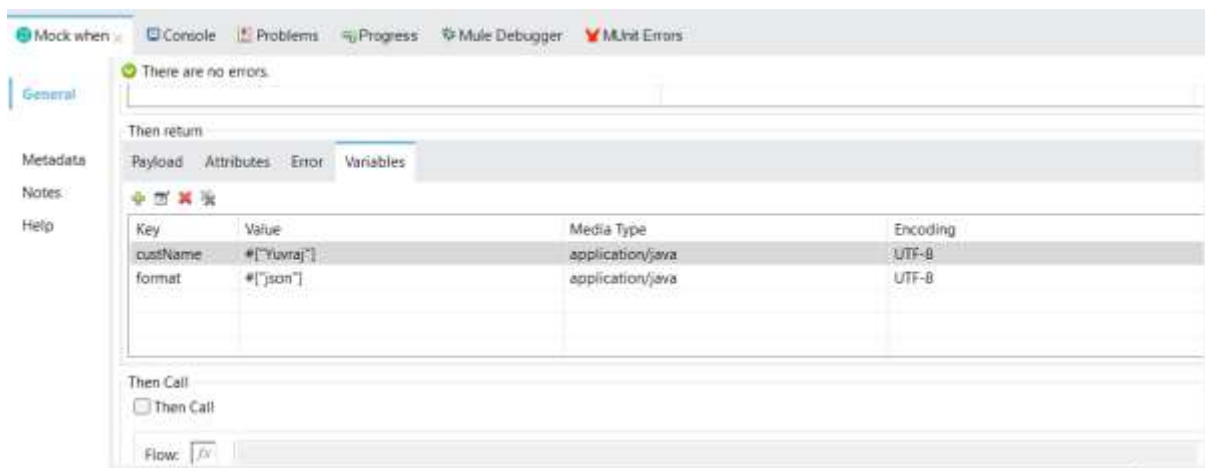
Now repeat the process and add the values below and click “OK”

Key = format

Value = #["json"]

Media Type = application/java (select from the drop down)

Encoding = UTF-8 (select from the drop down)



(“Then return” section is used to mock the event which is returned from the “fetching customer data” so set the values according to the component you are mocking)

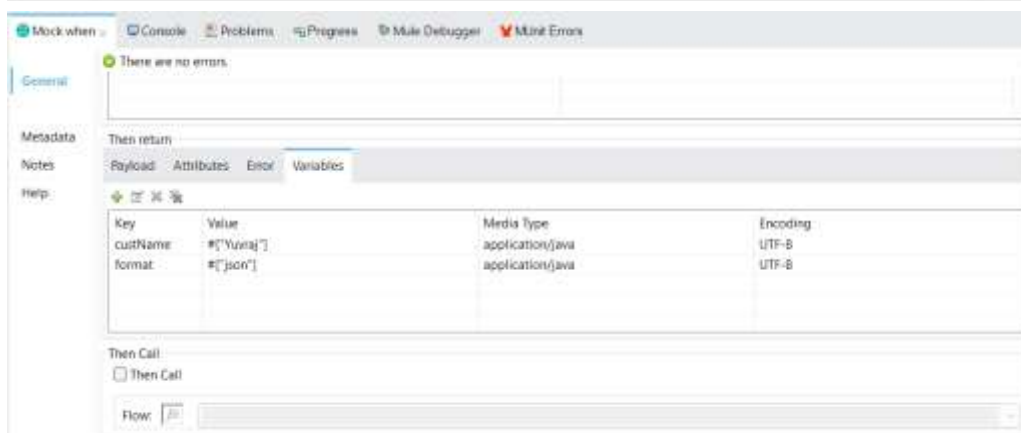
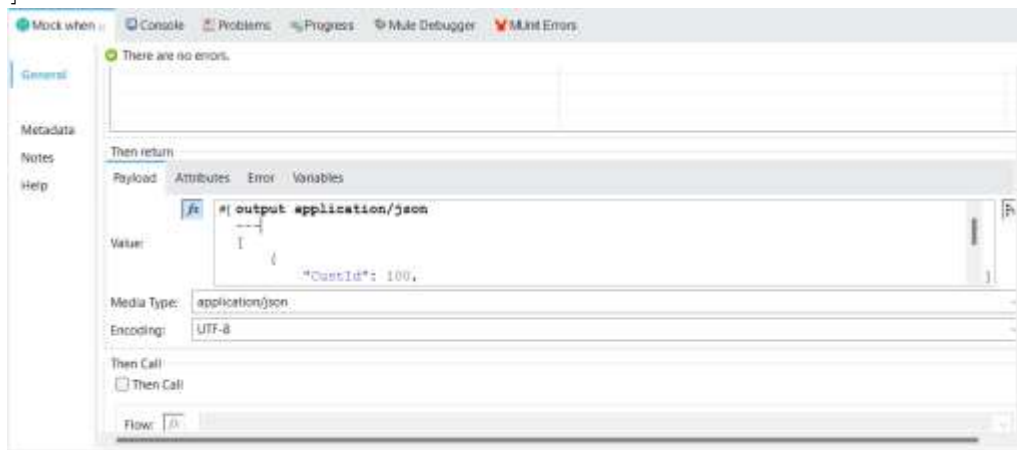
**Step 6:** Now we are going to mock the Flow reference “format-change-flow”

Repeat the above steps and we need to change the payload from the above steps, add the payload below and the rest steps are the same

**output application/json**

```

---
[
  {
    "CustId": 100,
    "OrderQty": "10",
    "CustName": "Yuvraj"
  }
]
  
```



Now the flow should look like below



©TGH Software Solutions Pvt. Ltd.

No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent

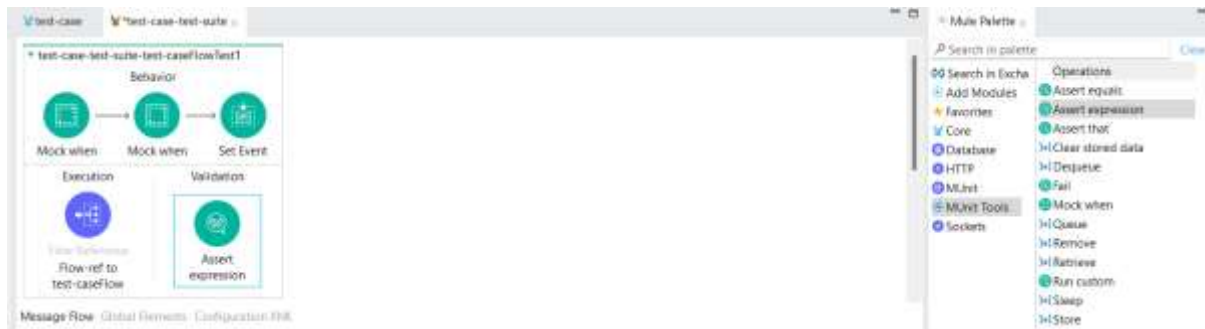


**Step 7:** Now we will Assert that the payload after invoking the “test-caseFlow” as [

```
{
  "CustId": 100,
  "OrderQty": "10",
  "CustName": "Yuvraj"
}
```

(we need to use assert expression because the output is an array object)

Now drag and drop an “Assert expression” on the “Validation” section of the flow



**Step 8:** In the expression section of “Assert expression” add

```
import * from dw::test::Asserts
---
payload must equalTo([
  {
    "CustId": 100,
    "OrderQty": "10",
    "CustName": "Yuvraj"
  }
])
```



(keep in mind to turn on the expression mode)



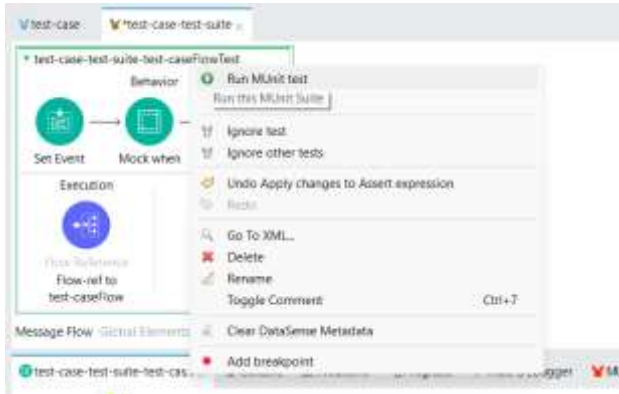
©TGH Software Solutions Pvt. Ltd.

No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent



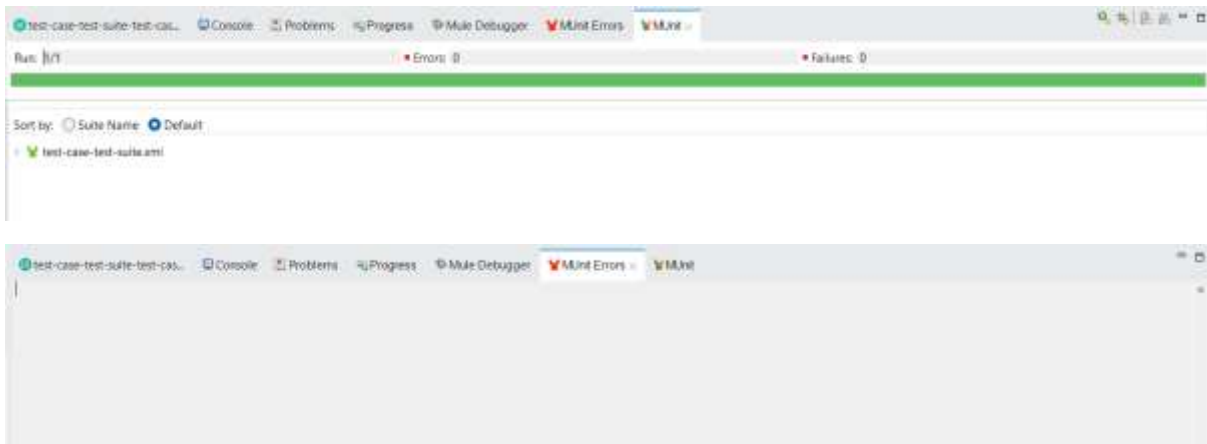
(feel free to change the output according to your flow)

**Step 9:** Now save the project and first select the flow then right-click on the flow



And click on “Run MUnit test”

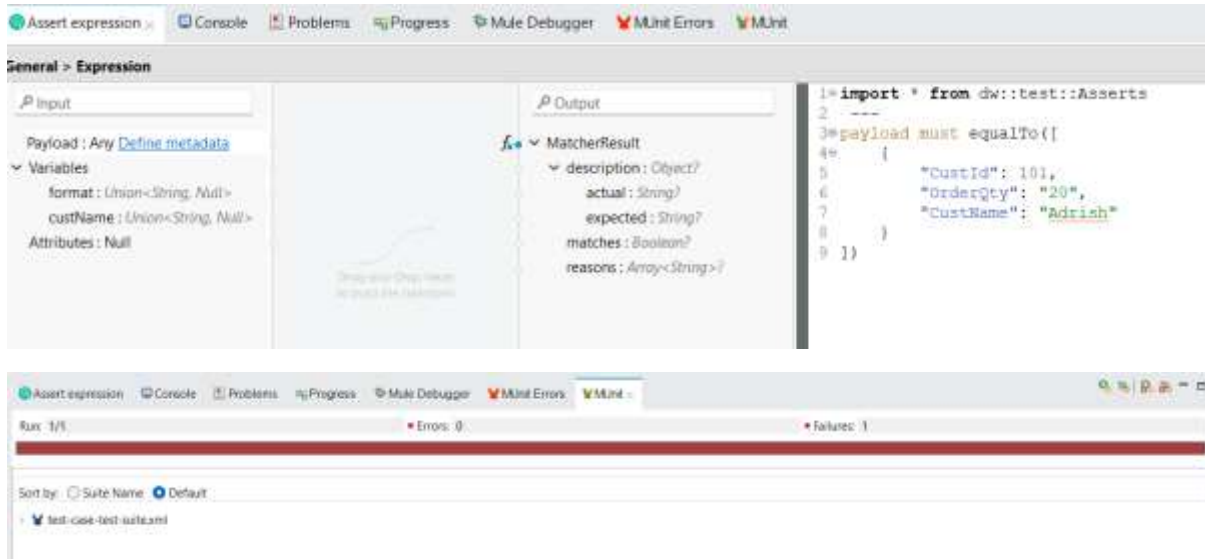
You should see that the test cases are passed with no errors



**Step 10:** Now go and change the value of “Expression” in “Assert Expression” to

```
import * from dw::test::Asserts
---
payload must equalTo([
  {
    "CustId": 101,
    "OrderQty": "20",
    "CustName": "Adrish"
  }
])
```

And again, run it. Now the test case will fail



The screenshot displays the MuleSoft IDE interface. The top toolbar includes icons for 'Assert expression', 'Console', 'Problems', 'Progress', 'Mule Debugger', 'MUnit Errors', and 'MUnit'. The main workspace is titled 'General > Expression' and is divided into three panes:

- Input:** Shows 'Payload: Any' with a 'Define metadata' link and 'Variables' section containing 'format: Union<String, Null>', 'custName: Union<String, Null>', and 'Attributes: Null'.
- Output:** Displays a 'MatcherResult' object with fields: 'description: Object?', 'actual: String?', 'expected: String?', 'matches: Boolean?', and 'reasons: Array<String?>'.
- Code Editor:** Contains the following Groovy code:

```
1= import * from dw::test::Asserts
2 ---
3= payload must equalTo([
4=   {
5=     "CustId": 101,
6=     "OrderQty": "20",
7=     "CustName": "Adrish"
8=   }
9 ])
```

Below the workspace, a status bar shows 'Run: 1/1', 'Errors: 0', and 'Failures: 1'. A table below the status bar shows the test results:

Sort by:	Suite Name	Default
	test-case-test-suite.xml	

## Writing functional test case

### Objective:

Now we are going to write a functional test case for the scenario

**Request:** <http://localhost:8081/testcase?custName=Adrish&format=json>



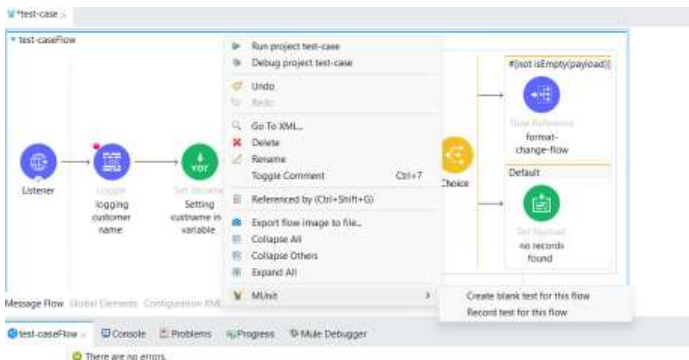
### Response:



### What is a functional test case?

A functional test case for verifying the behavior of a system typically follows a structured approach to ensure that the system functions correctly as per the requirements.

**Step1:** Click on the main flow that you want to write test cases (test-caseFlow) and then right-click on it. Select “MUnit” and then click “Create blank test for this flow”. This creates another blank test flow where you can write your test cases.



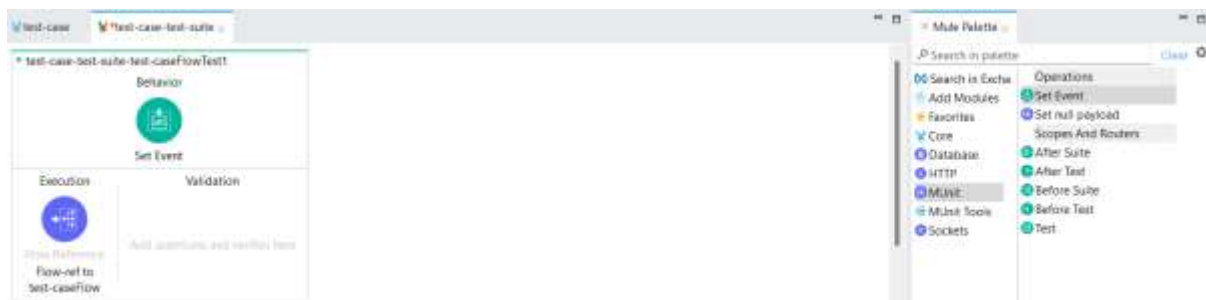
©TGH Software Solutions Pvt. Ltd.

No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent





**Step 2:** Drag and drop a set event processor from the mule pallet in the MUInt module onto the behaviour section.



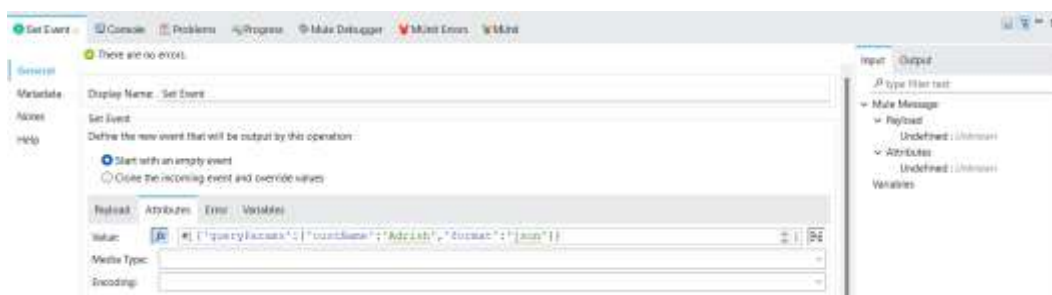
**Step 3:** Now we are going to create an event with an empty payload and attributes should contain 2 query parameters “custName” with value Adrish and format with value “json”.

Configure

```
{ 'queryParams': { 'custName': 'Adrish', 'format': 'json' } }
```

as the value for attributes in the set event.

(You can set the event according to the request you’re calling it with)



**Step 4:** Now we will Assert that the payload after invoking the “test-caseFlow” as [

```
{
  "CustId": 101,
  "OrderQty": "20",
  "CustName": "Adrish"
}
```

]

(we need to use assert expression because the output is an array object)

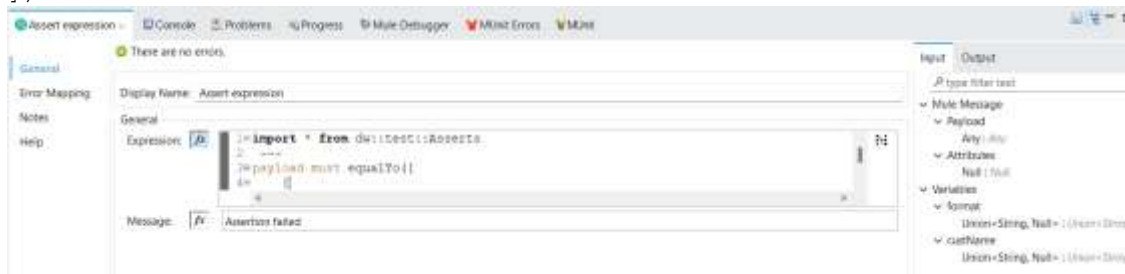
Now drag and drop an “Assert expression” on the “Validation” section of the flow

©TGH Software Solutions Pvt. Ltd.



**Step 5:** In the expression section of “Assert expression” add

```
import * from dw::test::Asserts
---
payload must equalTo([
  {
    "CustId": 101,
    "OrderQty": "20",
    "CustName": "Adrish"
  }
])
```

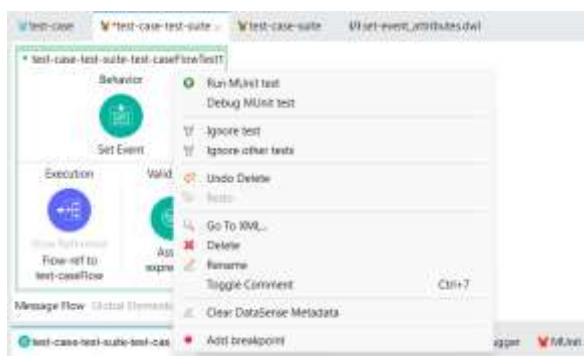


(keep in mind to turn on the expression mode)



(feel free to change the output according to your flow)

**Step 6:** Now save the project and first select the flow then right-click on the flow



©TGH Software Solutions Pvt. Ltd.

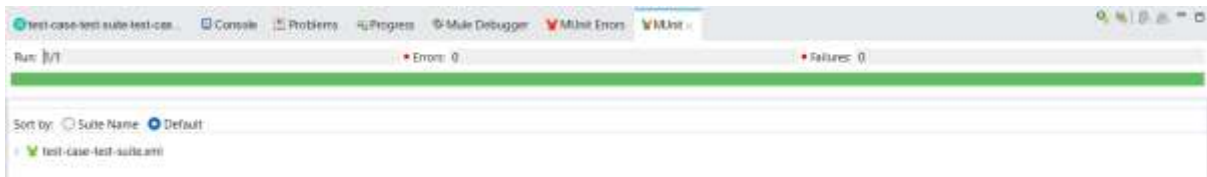
No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent



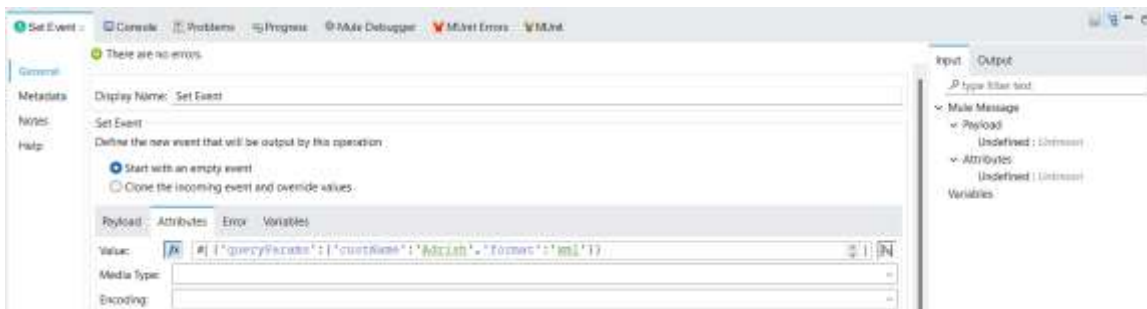


And click on “Run MUnit test”

You should see that the test cases are passed with no errors



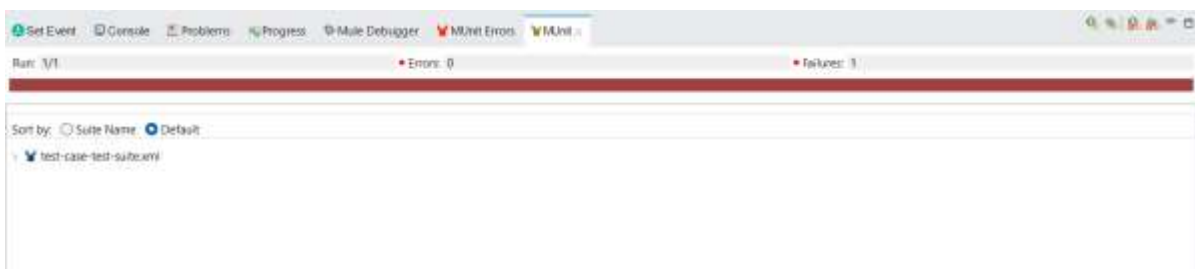
**Step 7:** Now go and change the query parameter format to “xml” in the attributes section of “Set Event”.



And again run the test. The test case will fail because the output given by the request we made does not match the expression ( {

```
{
  "CustId": 101,
  "OrderQty": "20",
  "CustName": "Adrish"
}
```

) We gave in the “Assert expression”



## Verify call

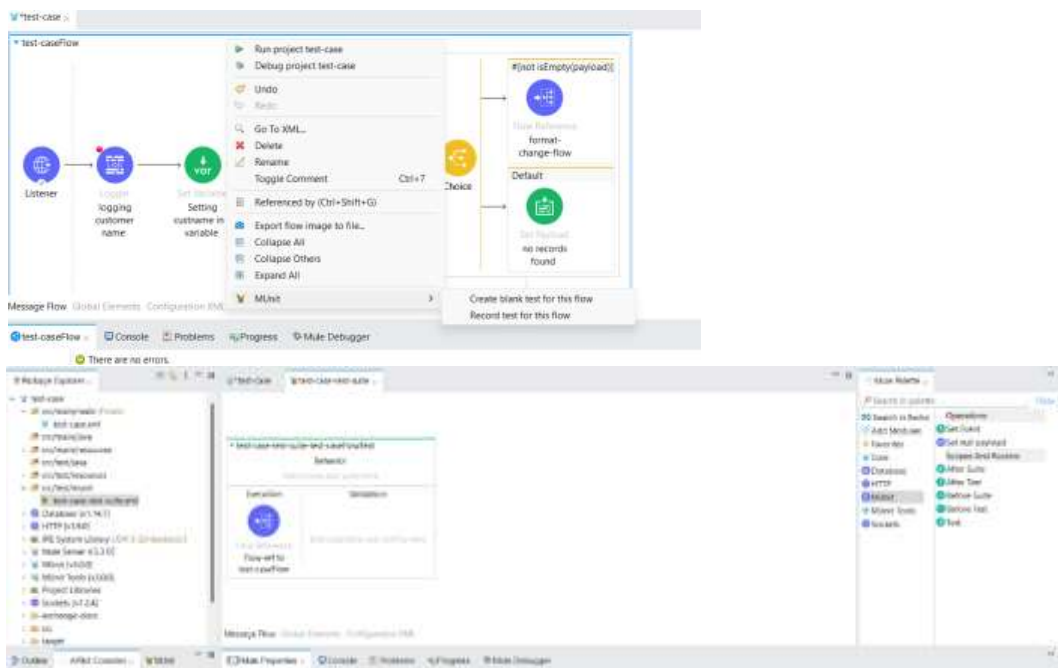
### What is the use of a verify call?

The Verify call processor allows you to verify if a processor was called.

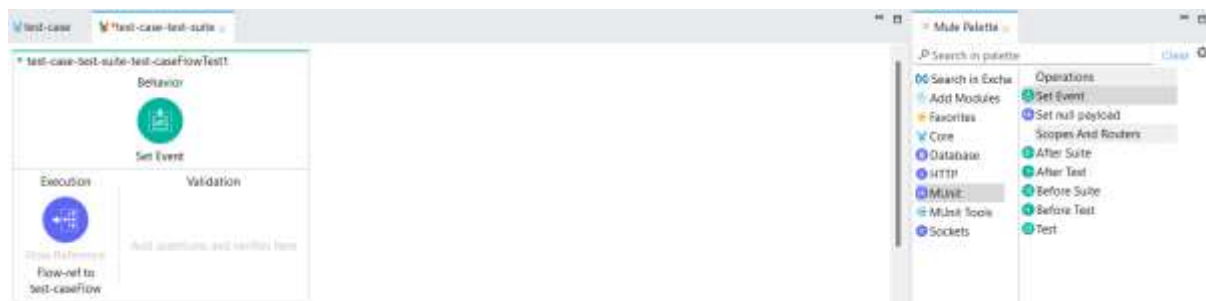
### Objective:

We will write another test case that will verify that “fetching customer data”(db:select) is called.

**Step 1:** Click on the main flow that you want to write test cases (test-caseFlow) and then right-click on it. Select “MUnit” and then click “Create blank test for this flow”. This creates another blank test flow where you can write your test cases.



**Step 2:** Drag and drop a set event processor from the mule pallet in the MUnit module onto the behaviour section.



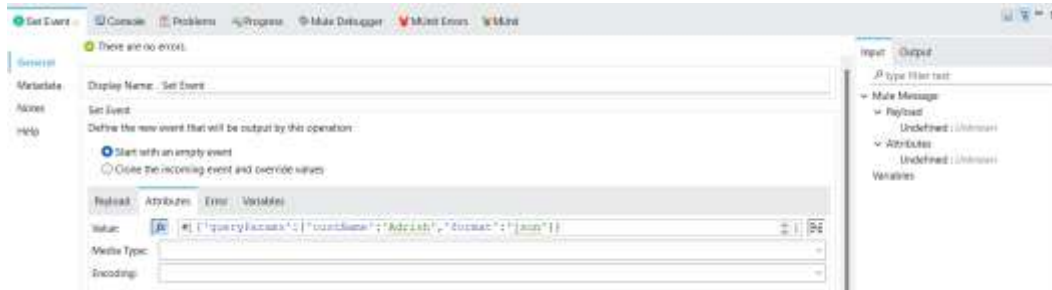
**Step 3:** Now we are going to create an event with an empty payload and attributes should contain 2 query parameters “custName” with value Adrish and format with value “json”.

## Configure

```
{ 'queryParams': { 'custName': 'Adrish', 'format': 'json' } }
```

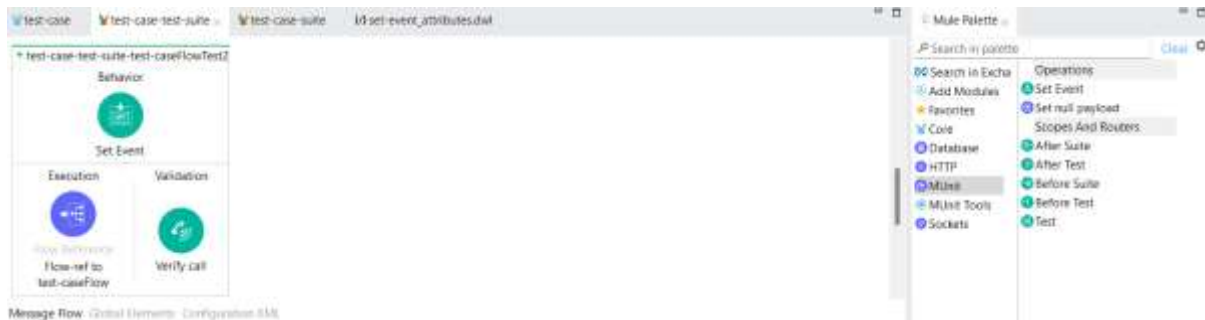
as the value for attributes in the set event.

(You can set the event according to the request you're calling it with)

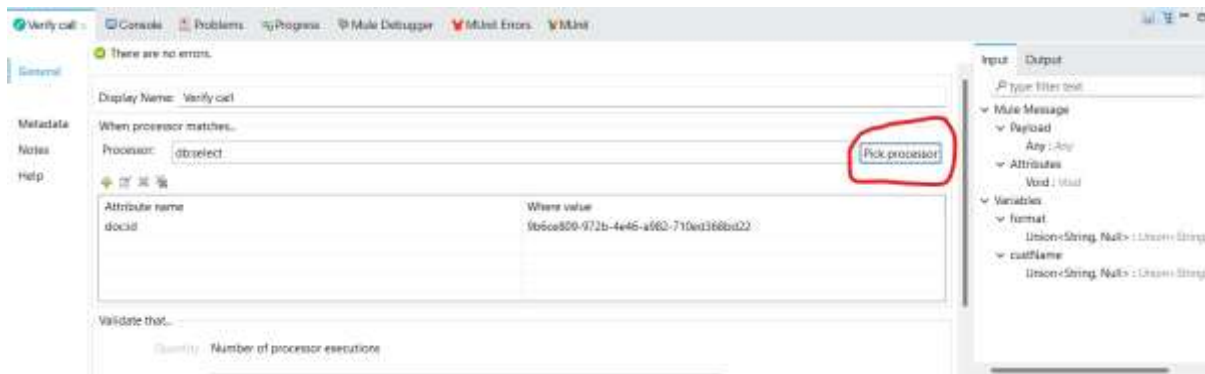


**Step 4:** Now we will use the “Verify call” and verify whether the “fetching customer data”(db:select) is called or not.

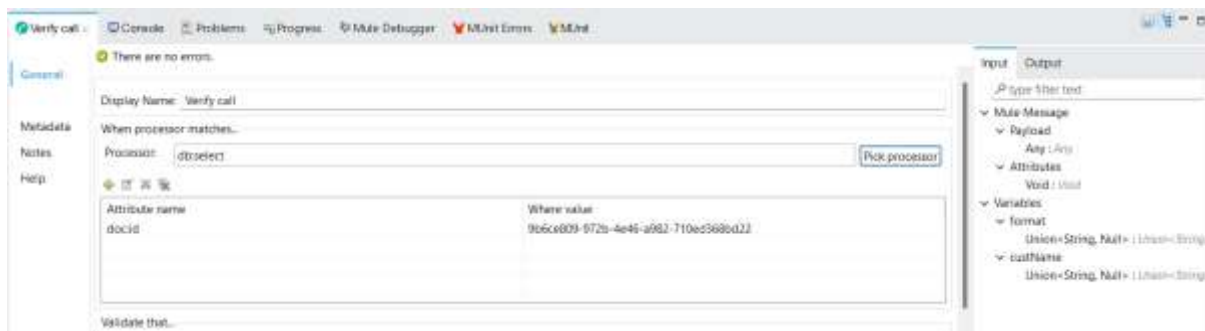
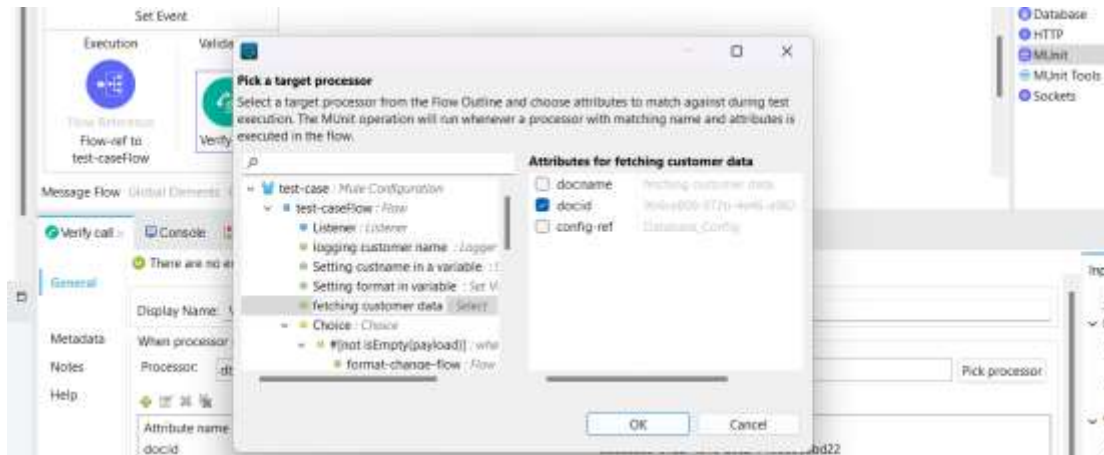
Now drag and drop a “Verify call” on the validation section of the flow



**Step 5:** Now click on the “Pick processor” button.

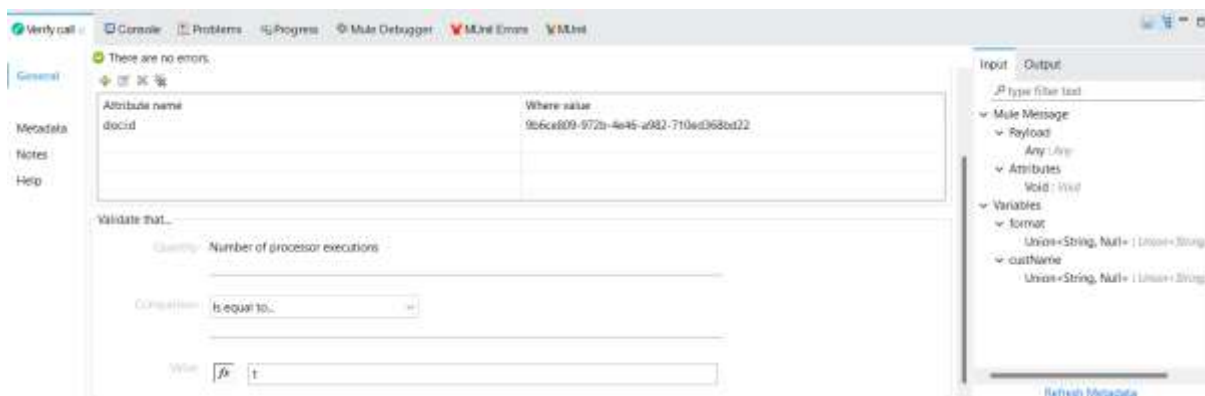


Now select “fetching customer data” and then select “doc:id” and click “Ok”

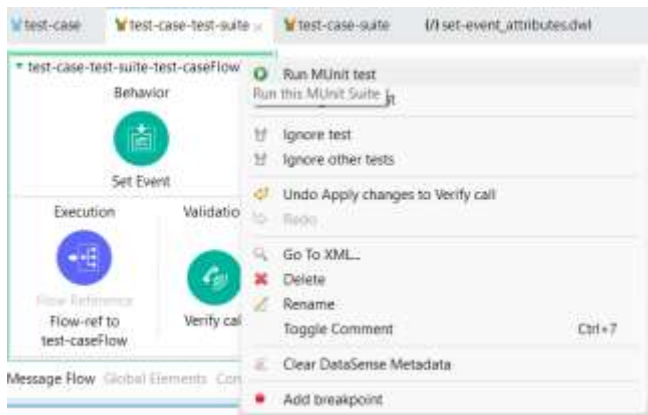


**Step 6:** Now we are going to configure the “Validate that” section (this section is used to validate how many times the processor we are verifying is executed)

Comparison = Is equal to.. (Choose from the drop down)  
Value = 1

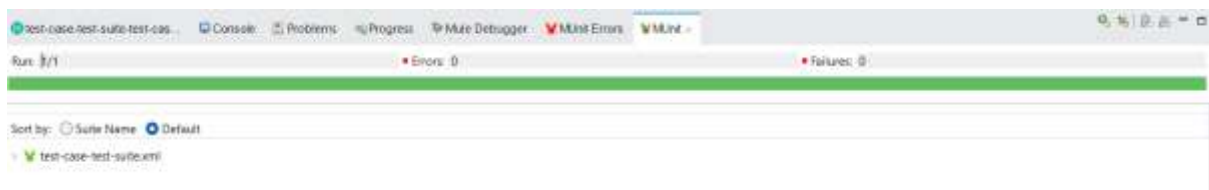


**Step 7:** Now save the project and first select the flow then right-click on the flow

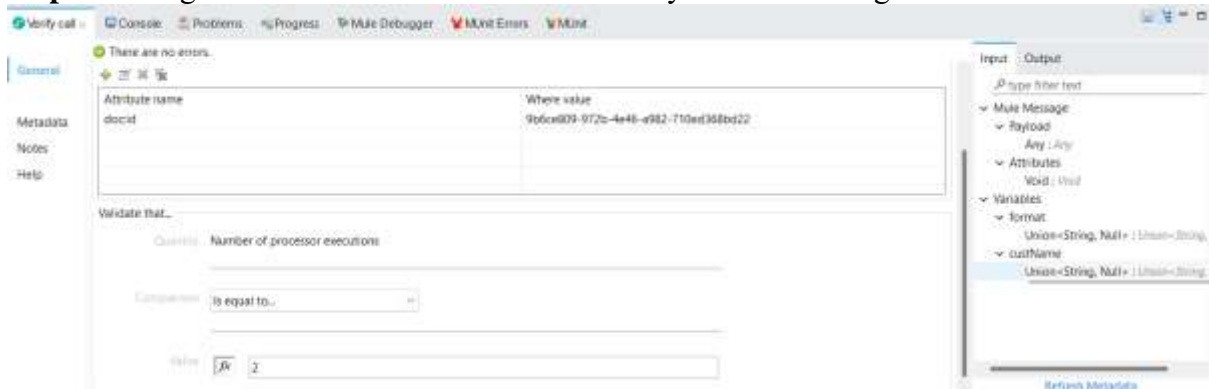


And click on “Run MUnit test”

You should see that the test cases are passed with no errors



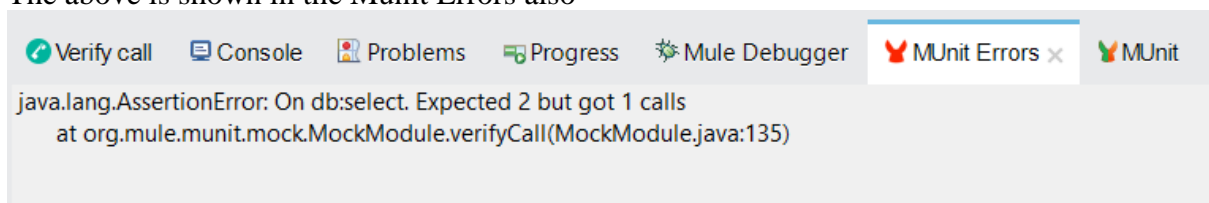
**Step 8:** Now go to “Validate that” section in “Verify call” and change the “value” to 2



And again, run the project. Now the test will fail because the number of times the “fetching customer data” is executed is 1 and we have given 2.



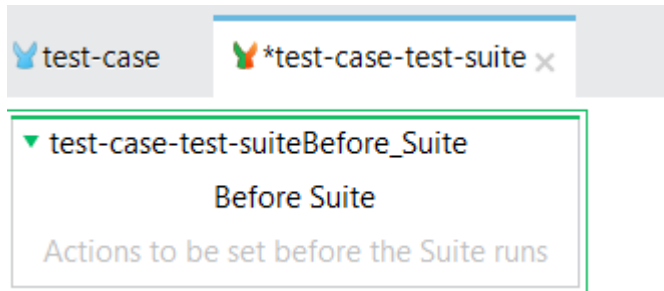
The above is shown in the Munit Errors also



## Before/After Suite & Before/After Test

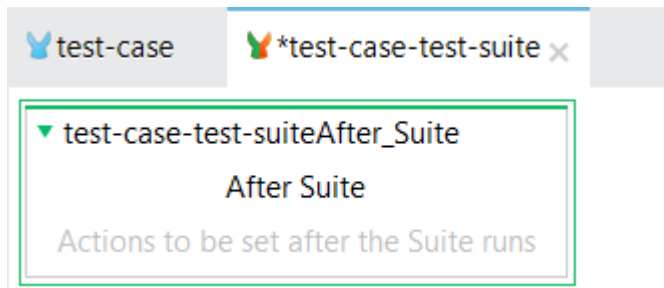
### Before Suite Scope

The MUnit Before Suite contains code that is meant to be executed before the whole suite. The Before Suite has one execution rule: Run before all the tests, just once. For example, suppose you have a MUnit Test Suite file with four tests. The code inside an MUnit Before Suite runs just once, before your four tests.



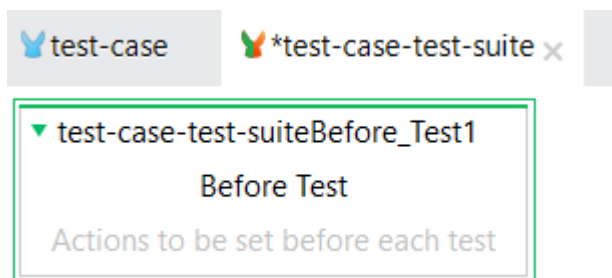
### After Suite Scope

The MUnit After Suite is a scope. It can contain code that is meant to be executed after the whole suite. The After suite has one execution rule: Run after all the tests, just once. For instance, let's suppose you have a MUnit Test Suite File with four tests. The code inside an MUnit After Suite, runs just once, after all your tests.



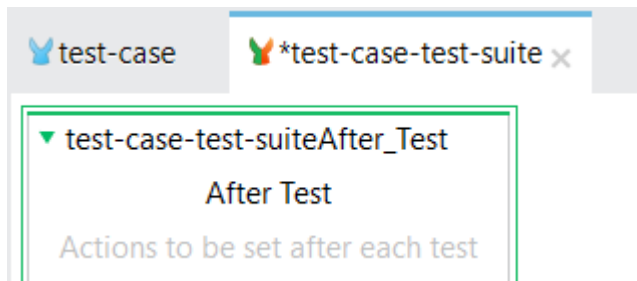
### Before Test Scope

The MUnit Before Test Scope contains code that is meant to be executed before each test. Each Before Test scope follows the same execution rule: Run before each test. For instance, let's suppose you have a MUnit Test Suite file with four tests. The code inside an MUnit Before test runs before each of your four tests; it runs four times.



## After Test Scope

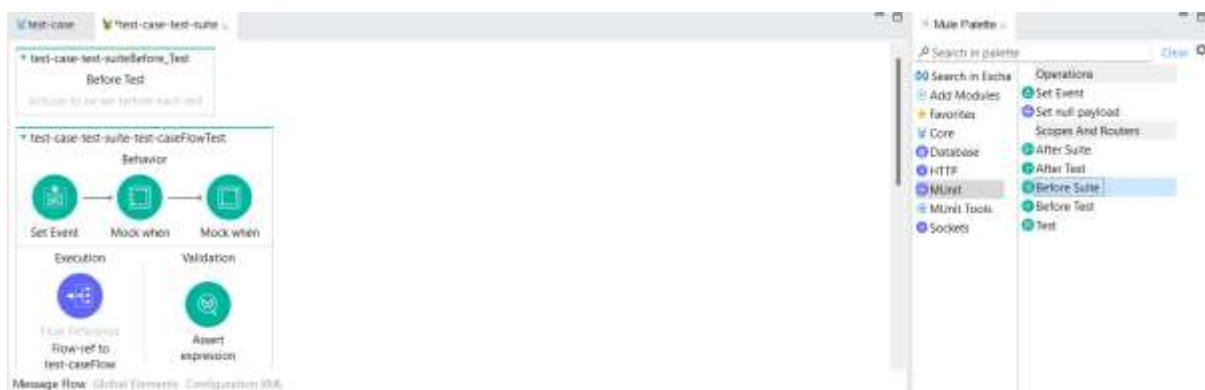
The MUnit After Test Scope contains code that is meant to be executed after each test. Each After Test scope follows the same execution rule: Run after each test. For instance, let's suppose you have a MUnit Test Suite file with four tests. The code inside an MUnit After Test runs after each of your four tests; it runs four times.



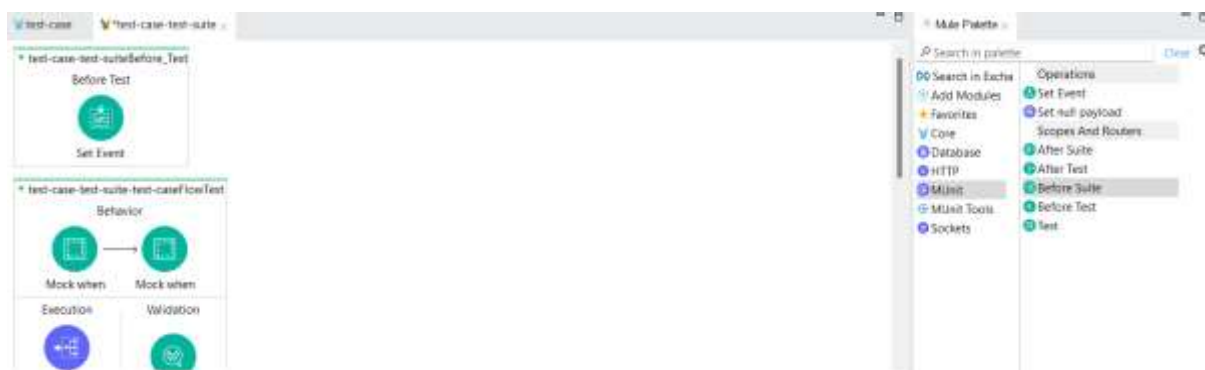
### Objective:

We are going to create a scenario where we will have the same events for all the test cases. We are going to execute the “Set event” in the “Before suite” scope so that it can be utilized by all the test cases and executed once.

**Step 1:** Drag and drop a “Before suite” scope on the test suite.



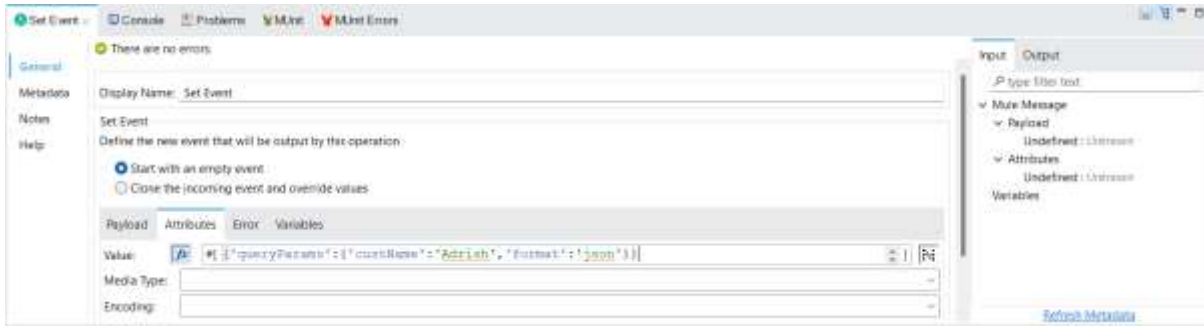
**Step 2:** Now drag and drop a “Set event” processor on the “Before suite” scope.



**Step 3:** Now configure the attributes section of the “Set event” processor with these values

```
{'queryParams':{'custName':'Adrish','format':'json'}}
```

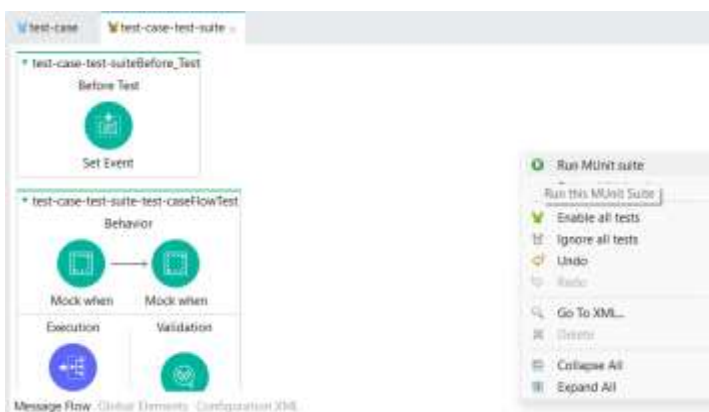
And leave the rest empty



**Step 4:** Now delete all the set events from the test cases



**Step 5:** Save the project then right-click and click on “Run MUnit suite”






You will see that the functional test case and verify call test case are passing but the Unit Test case will fail

test-case
test-case-test-suite ×


test-case-test-suite-test-caseFlowTest

Behavior





Mock when

→





Mock when

<p style="text-align: center; margin: 0;">Execution</p> <div style="text-align: center; margin-bottom: 5px;">  </div> <p style="text-align: center; margin: 0;">Flow Reference Flow-ref to test-caseFlow</p>	<p style="text-align: center; margin: 0;">Validation</p> <div style="text-align: center; margin-bottom: 5px;">  </div> <p style="text-align: center; margin: 0;">Assert expression</p>
---	---

test-case-test-suite-test-caseFlowTest1

Behavior



Add mocks and spies here

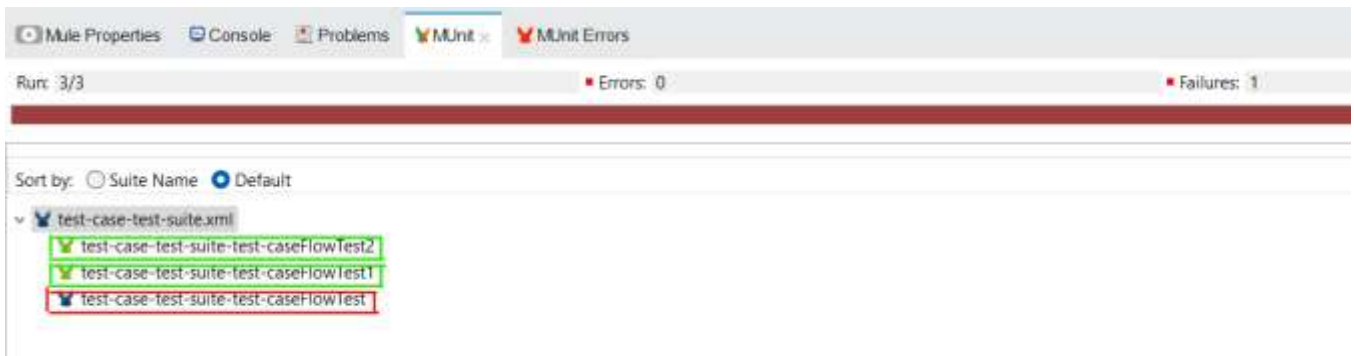
<p style="text-align: center; margin: 0;">Execution</p> <div style="text-align: center; margin-bottom: 5px;">  </div> <p style="text-align: center; margin: 0;">Flow Reference Flow-ref to test-caseFlow</p>	<p style="text-align: center; margin: 0;">Validation</p> <div style="text-align: center; margin-bottom: 5px;">  </div> <p style="text-align: center; margin: 0;">Assert expression</p>
---	---

test-case-test-suite-test-caseFlowTest2

Behavior

Add mocks and spies here

<p style="text-align: center; margin: 0;">Execution</p> <div style="text-align: center; margin-bottom: 5px;">  </div> <p style="text-align: center; margin: 0;">Flow Reference Flow-ref to</p>	<p style="text-align: center; margin: 0;">Validation</p> <div style="text-align: center; margin-bottom: 5px;">  </div> <p style="text-align: center; margin: 0;">Verify call</p>
---	---

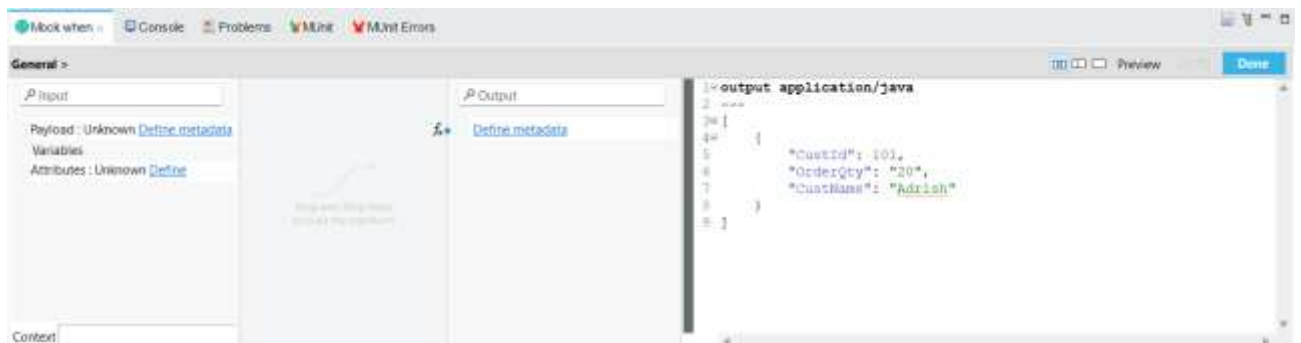


**Step 6:** To make the test cases pass we need to change both the “Mock when” payload and variables data.

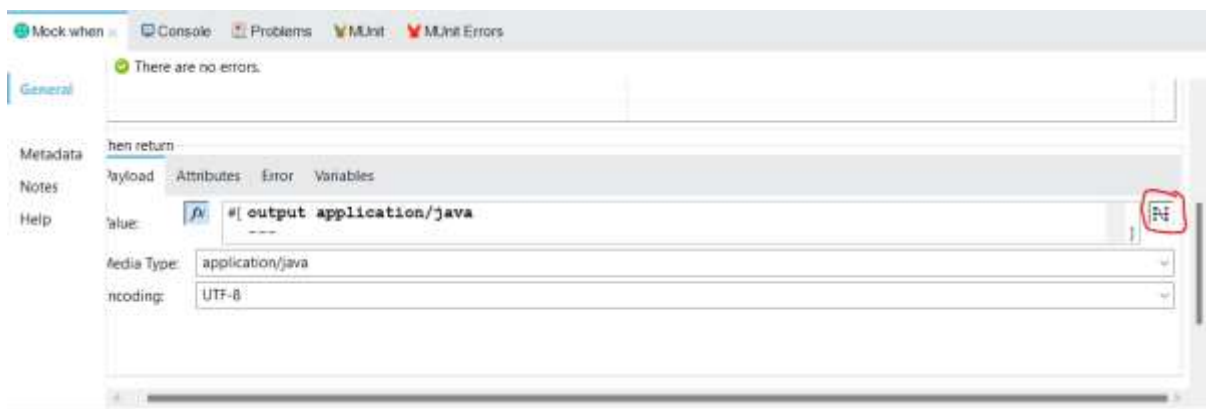
Change the values by referring to the screenshots below

The “Mock when” which is mocking the db:select change the payload to

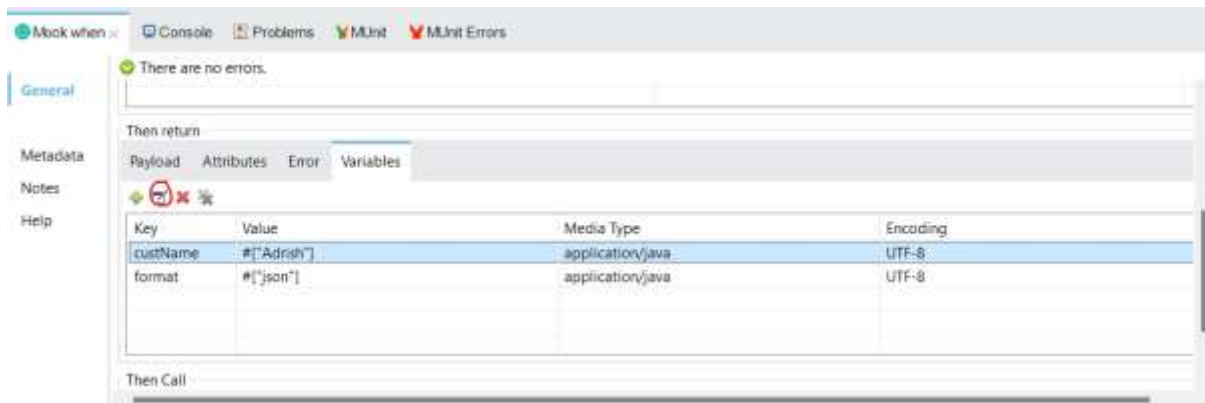
```
[
  {
    "CustId": 101,
    "OrderQty": "20",
    "CustName": "Adrish"
  }
]
```



To go in expression mode click the button highlighted in red in the below screenshot.



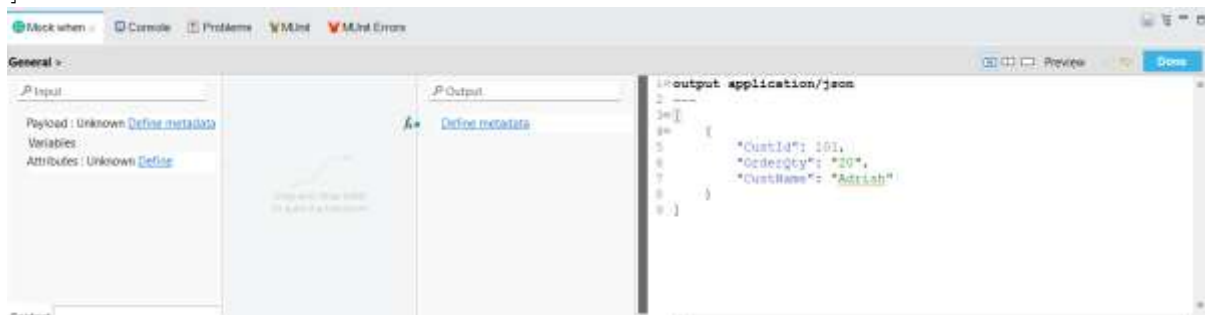
And variables custname = #["Adrish"] and keep format as json.



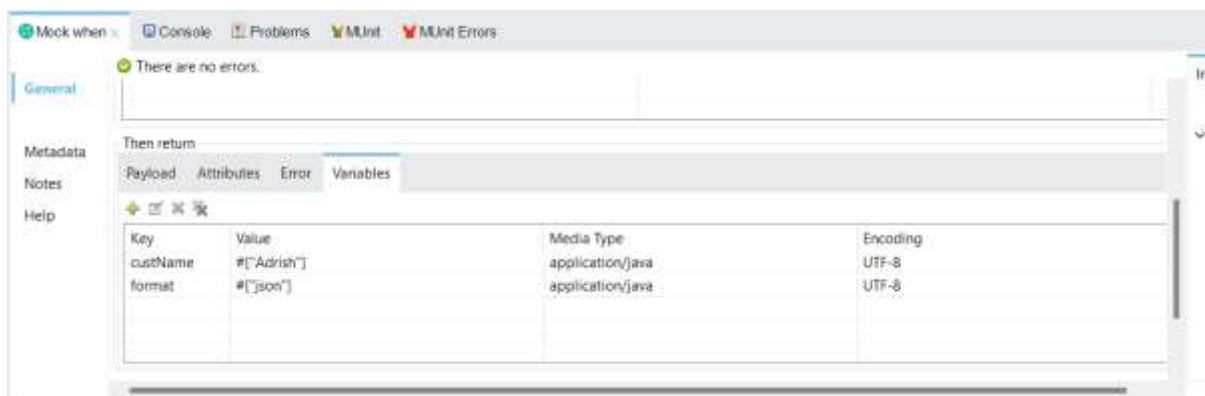
Click the button highlighted in red to edit the variables.

Now change the payload of the “Mock when” which is mocking “the flow-ref”.

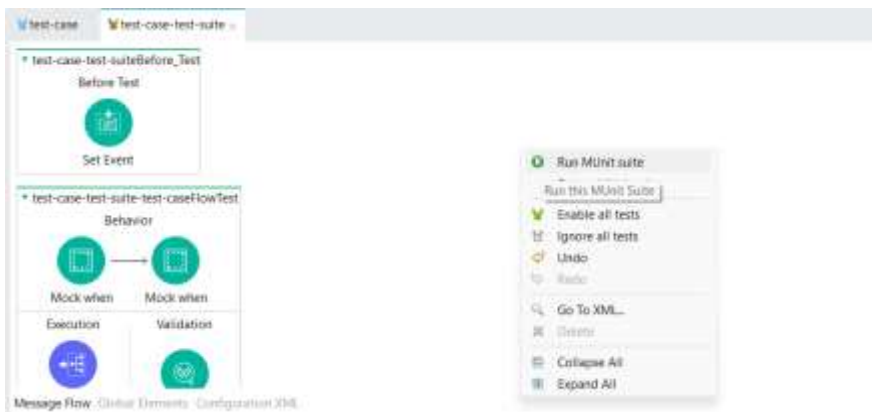
```
[
  {
    "CustId": 101,
    "OrderQty": "20",
    "CustName": "Adrish"
  }
]
```



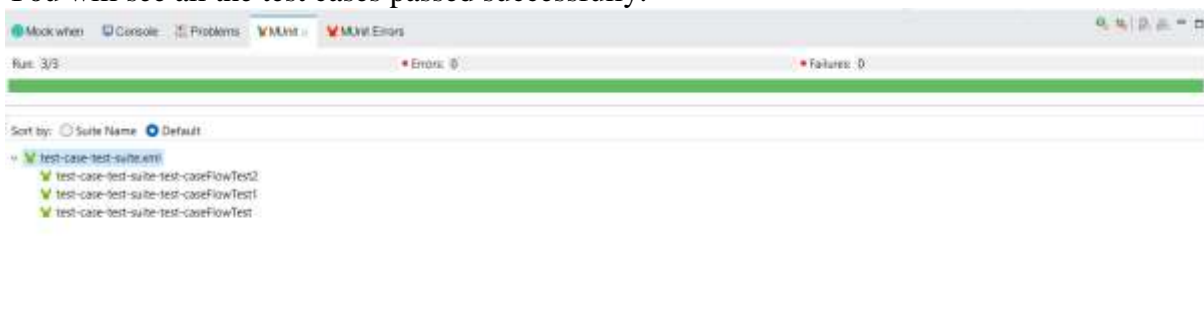
And variables custname = #["Adrish"] and keep the format as json.



**Step 7:** Now save the project then right-click and click on “Run MUnit suite”



You will see all the test cases passed successfully.



# Spy Event Processor

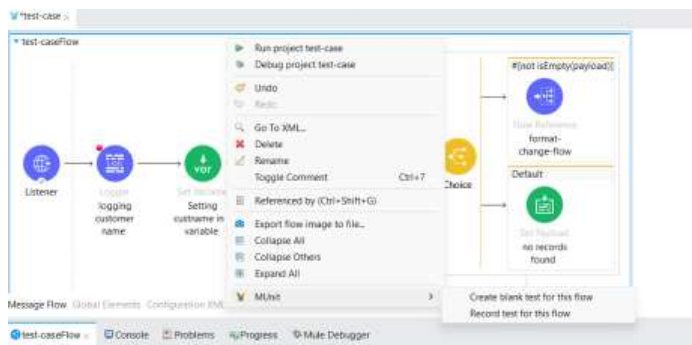
## What is the use of Spy Event Processor?

The “Spy Processor” enables you to spy on what happens before and after an event processor is called. This enables you to validate, for example, that a selected Mule Event reaches a specific event processor containing a specific payload or variable. Setting a spy processor tells MUnit to run a set of instructions (usually assertions or verifications) before and/or after the execution of the spied event processor.

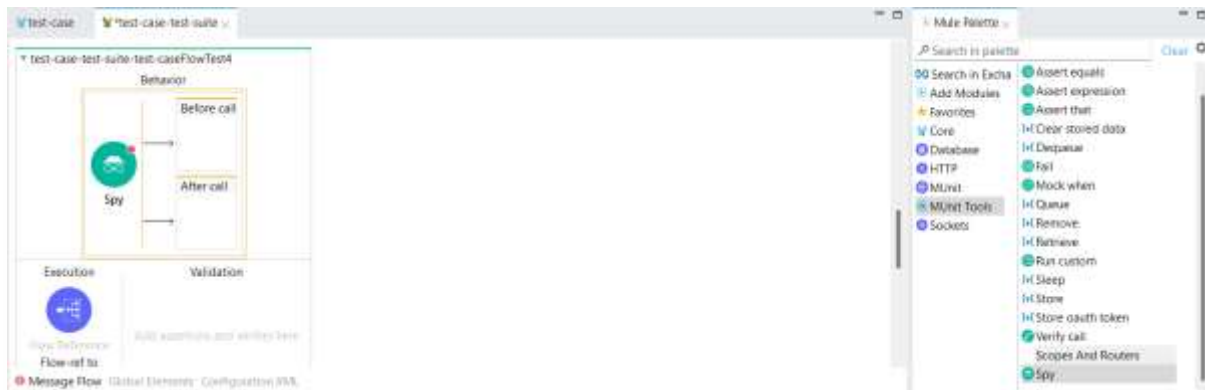
## Objective:

We are going to spy “fetching customer data”

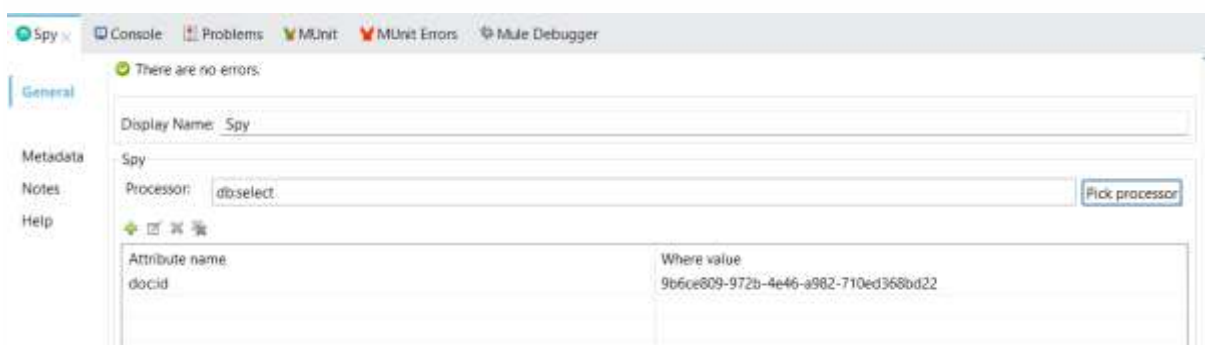
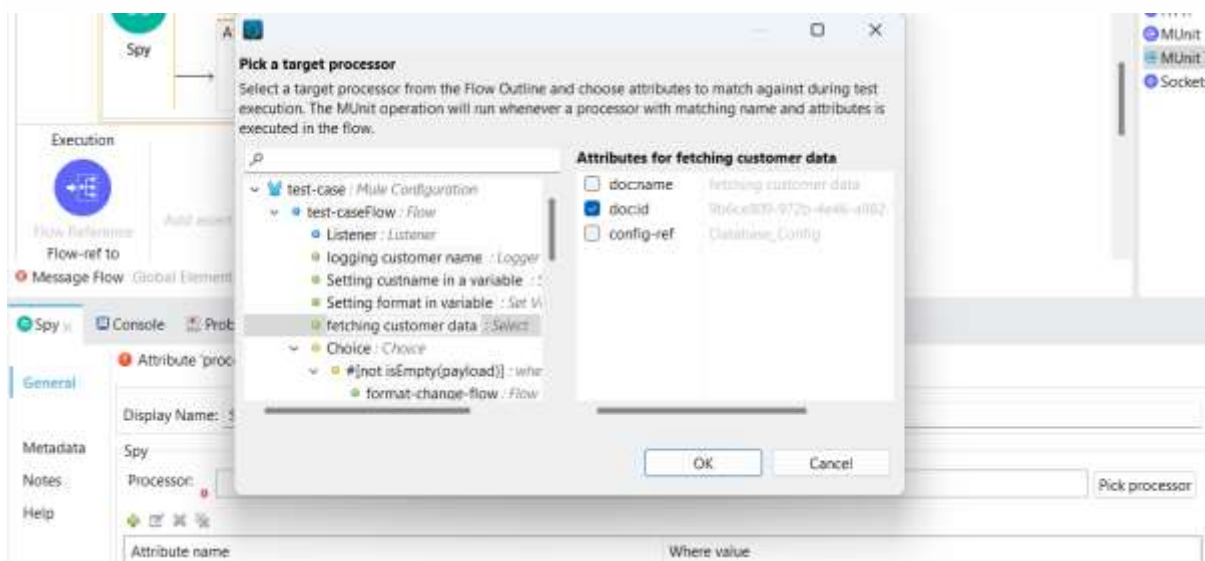
**Step 1:** Click on the main flow that you want to write test cases (test-caseFlow) and then right-click on it. Select “MUnit” and then click “Create blank test for this flow”. This creates another blank test flow where you can write your test cases.



**Step 2:** drag and drop the “Spy Event Processor” from the “MUnit Tools” module on the behavior section of the test flow.



**Step 3:** Now click on the pick processor button and select the “fetching customer data” (db:select) and then select “doc:id” and click on “OK”.



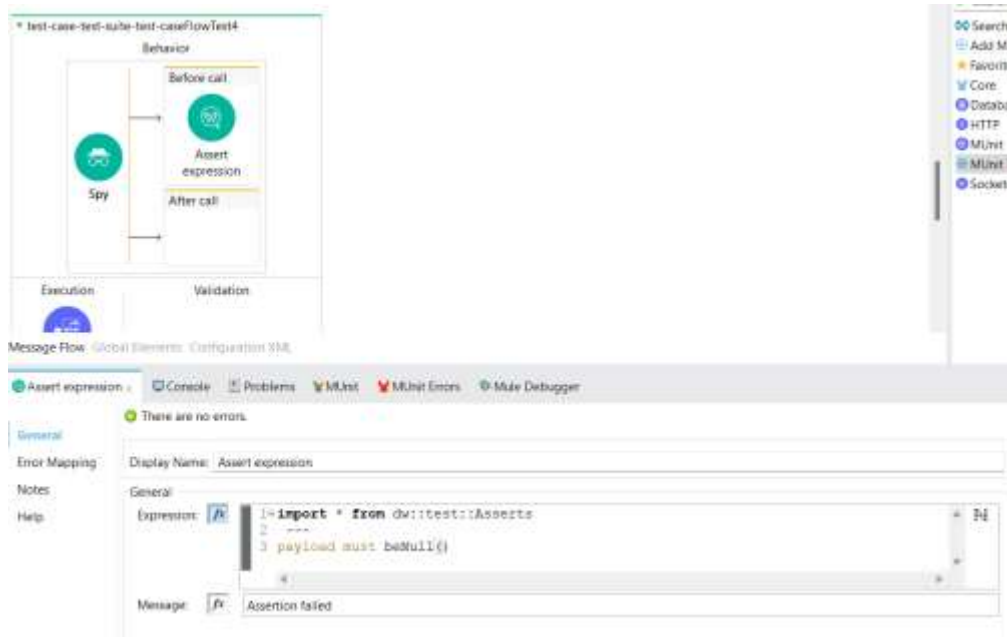
**Step 4:** Now drag and drop an assert expression on the “Before call” section of the “Spy Event Processor” and configure its “Expression” as

```
import * from dw::test::Asserts
---
payload must beNull()
```

©TGH Software Solutions Pvt. Ltd.

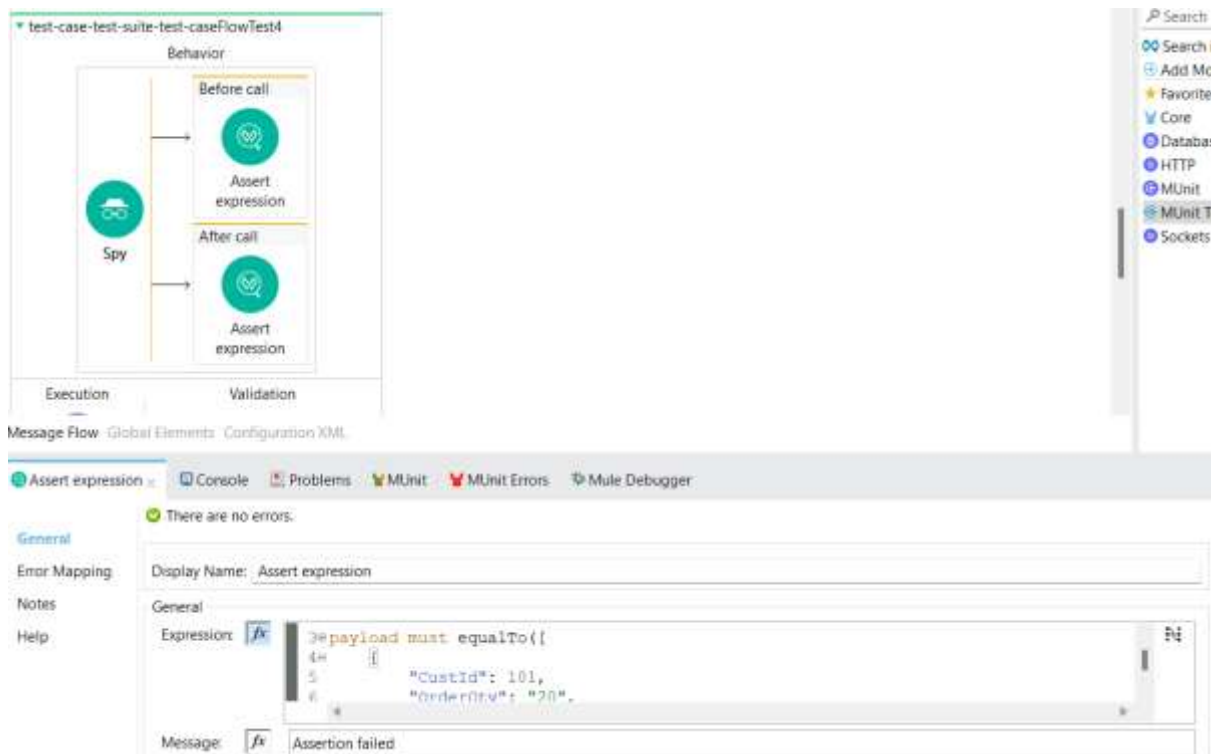
No part of this document may be copied, reproduced, republished, uploaded, posted, publicly displayed, encoded, translated, transmitted or distributed in any way to any other computer, server, website or other medium for publication or distribution, without TGH's prior written consent





**Step 5:** Now drag and drop another “Assert Expression” on the “After call” section of the “Spy Event Processor” and configure its “Expression” as

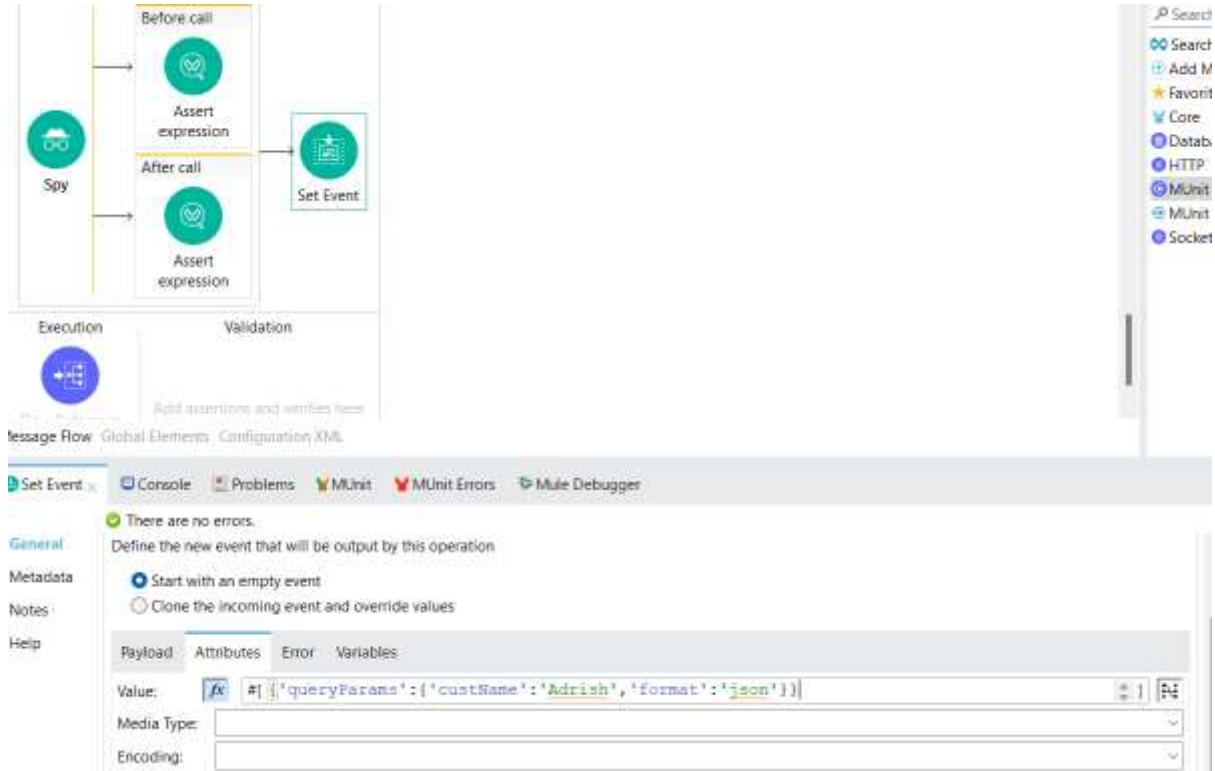
```
import * from dw::test::Asserts
---
payload must equalTo([
  {
    "CustId": 101,
    "OrderQty": "20",
    "CustName": "Adrish"
  }
])
```



©TGH Software Solutions Pvt. Ltd.

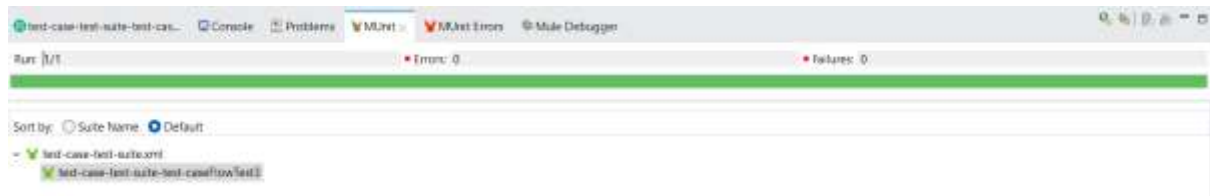
**Step 6:** Now drag and drop “Set Event” processor from the “MUnit” module on the behavior section after the “Spy Event Processors”. And set the attributes as

```
{'queryParams':{'custName':'Adrish','format':'json'}}
```



The screenshot shows the MUnit configuration window for a 'Set Event' processor. The 'Payload' tab is active, showing the JSON payload: `{ "queryParams": { "custName": "Adrish", "format": "json" } }`. The 'Media Type' is set to 'application/json'. The 'Error' and 'Variables' tabs are also visible. In the background, the flow diagram shows a 'Spy' processor followed by 'Before call' and 'After call' sections, each containing an 'Assert expression' processor, and finally a 'Set Event' processor.

**Step 6:** Now save the project and first select the flow then right-click on the flow and click on “Run MUnit test”. The test case will run successfully.



The screenshot shows the MUnit test results window. The test run is successful, with 0 errors and 0 failures. The test case is 'test-case-test-suite-test-case/flow/test3'. The results are displayed in a table with columns for 'Suite Name' and 'Default'.

**Step 7:** Now just switch the position of “Set event” and “Spy Event Processor”

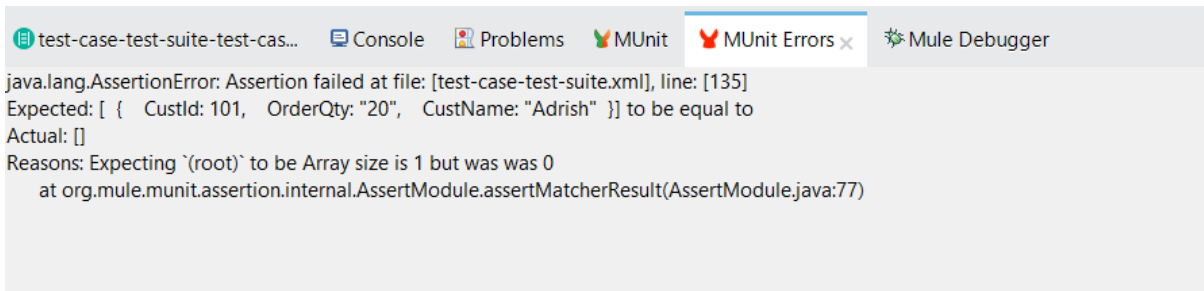




**Step 8:** Now save the project and first select the flow then right-click on the flow and click on “Run MUnit test”. The test case will fail.



With an error



(This is a Bug in “Spy Event Processor” It changes the mule message to null (payload and attributes). It is best to keep the “Spy Event Processor” at first and then other processors. This may be fixed on further updates so it will depend on the version of anypoint studio if you face the bug or not)



# TGH

Making Integrations Simpler

## TGH Software Solutions Pvt. Ltd.

[www.techygeekhub.com](http://www.techygeekhub.com)

At TGH, we specialize in driving digital transformation through seamless Integration Technologies.

Operating as an INTEGRATION FACTORY, we serve as a one-stop shop for all your integration needs. Our expert team is well-versed in enterprise software and legacy system integration, along with leading iPaaS technologies like Boomi, MuleSoft, Workato, OIC, and more.

We're committed to enhancing business processes and solving problems through our integration expertise.



### Email address

[connect@techygeekhub.com](mailto:connect@techygeekhub.com)



### Phone number

+ 011-40071137  
+ 91-8810610395



### Our offices

#### Noida Office

iThum  
Plot No -40, Tower A,  
Office No: 712,  
Sector-62, Noida,  
Uttar Pradesh, 201301

#### Hyderabad Office

Plot no: 6/3, 5th Floor,  
Techno Pearl Building,  
HUDA Techno Enclave,  
HITEC City, Hyderabad,  
Telangana 500081

