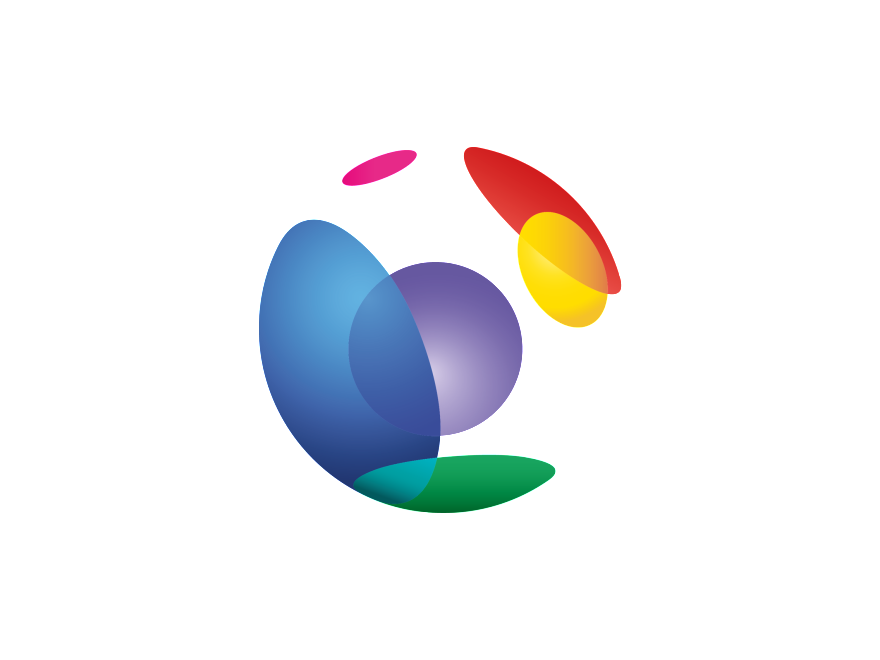
****

**April 2018**

**Daisy Proof of Concept (PoC)**

**User Guide**

# Revision History

|  |  |  |
| --- | --- | --- |
| **Rev Number** | **Date** | **Summary of changes** |
| v 0.1 | 30th April, 2018 | Created the initial draft |
|  |  |  |
|  |  |  |

# Tech Mahindra Contacts

|  |  |  |  |
| --- | --- | --- | --- |
| **Company** | **Location** | **Name** | **Role & Contact** |
| Tech Mahindra | Offshore | Sathya Loganathan Shanmugavel | DevTest SV Consultant  [SL00484554@TechMahindra.com](mailto:SL00484554@TechMahindra.com) |
| Tech Mahindra | Offshore | Anjali Rai | DevTest SV Consultant  [AR00353866@TechMahindra.com](mailto:AR00353866@TechMahindra.com) |

Table of Contents

[How to Trigger XML from DevTest and Validate Responses 4](#_Toc513465561)

[Steps to generate request XML from SV 4](#_Toc513465562)

[To Capture Response 8](#_Toc513465563)

[Replenishment Process 12](#_Toc513465564)

[Validating OUTRPL notification and automating INRPL request 13](#_Toc513465565)

# How to Trigger XML from DevTest and Validate Responses

1. **Workflow:**

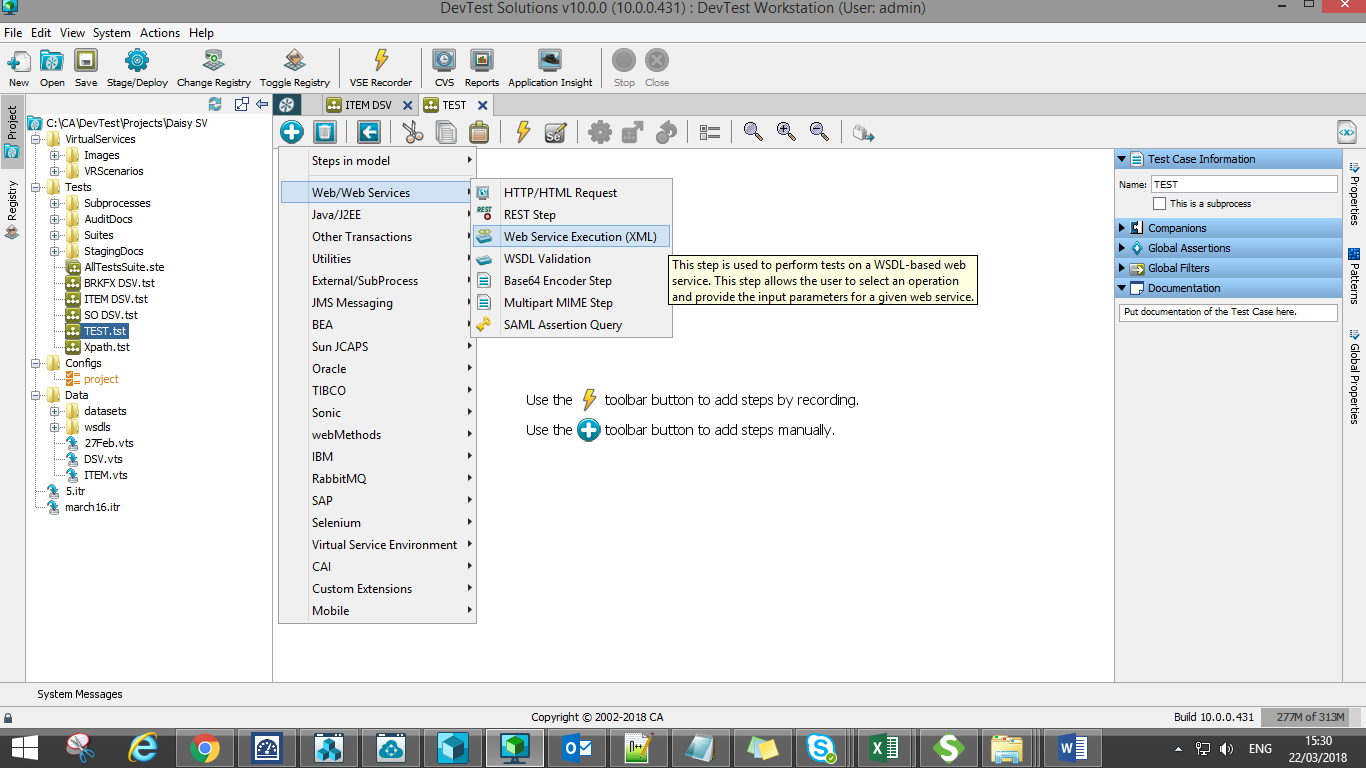
|  |
| --- |
| **General Flow: DAISY > PAL > SC UAF > WMS** |
| **Virtual Flow: VS > PAL > SC UAF > WMS** |

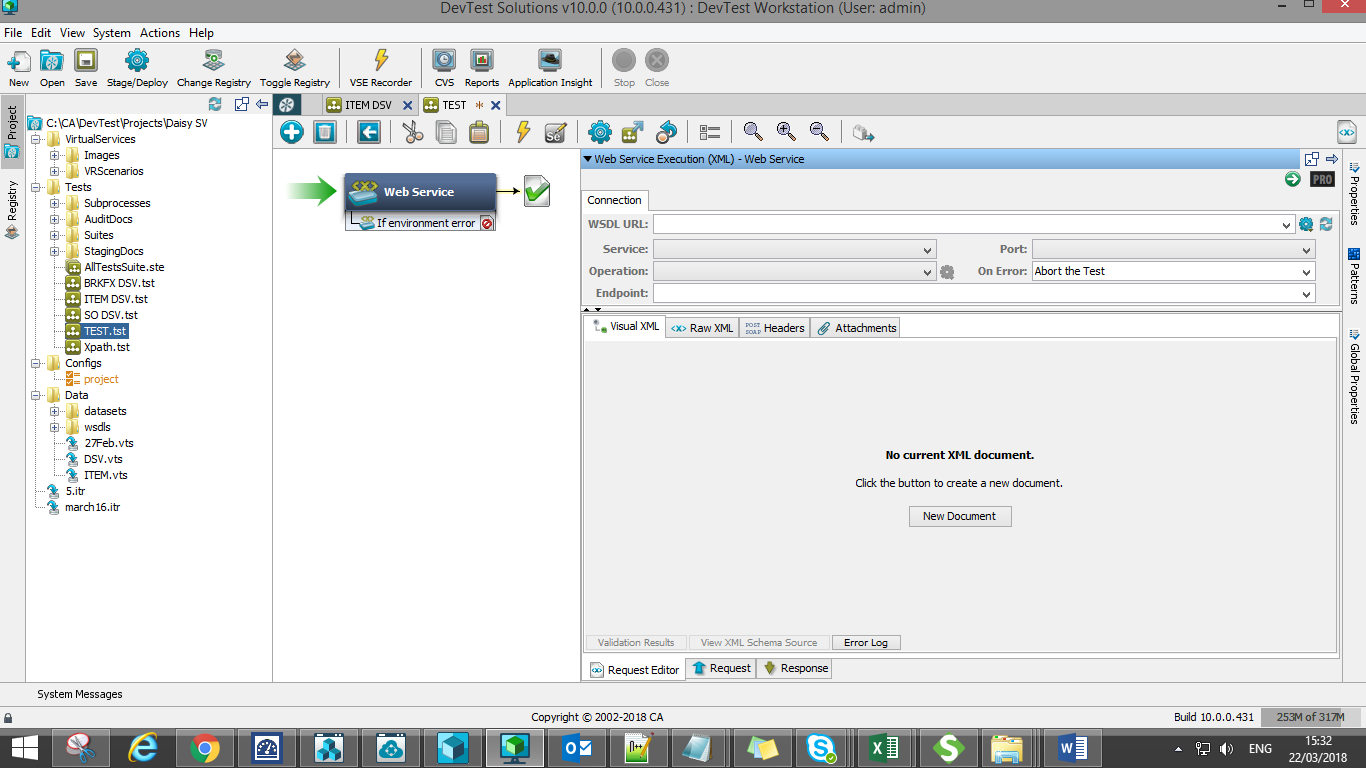
1. **Plan:**

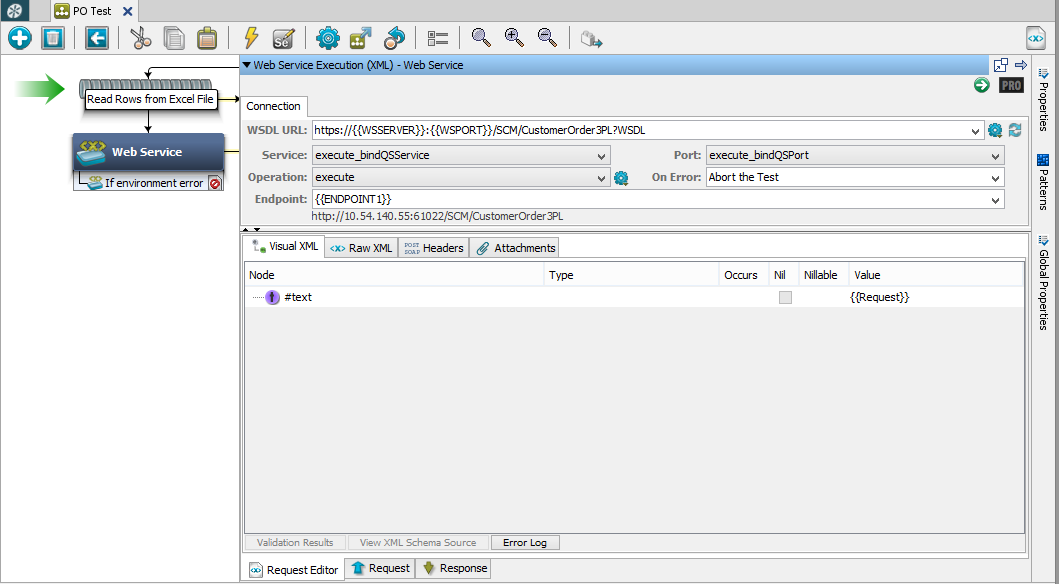
To generate XML (fire xml) in the absence of DAISY, from SV, first create a web service TEST CASE using the wsdl details shared by PAL team and add step to read data from data sheet for the N no of requests to be fired at a time from SV to PAL and also add Do Nothing step to compare the result that comes from PAL. Thus we fired xml’s in the place of DAISY and validated the response received from PAL in the absence of DAISY

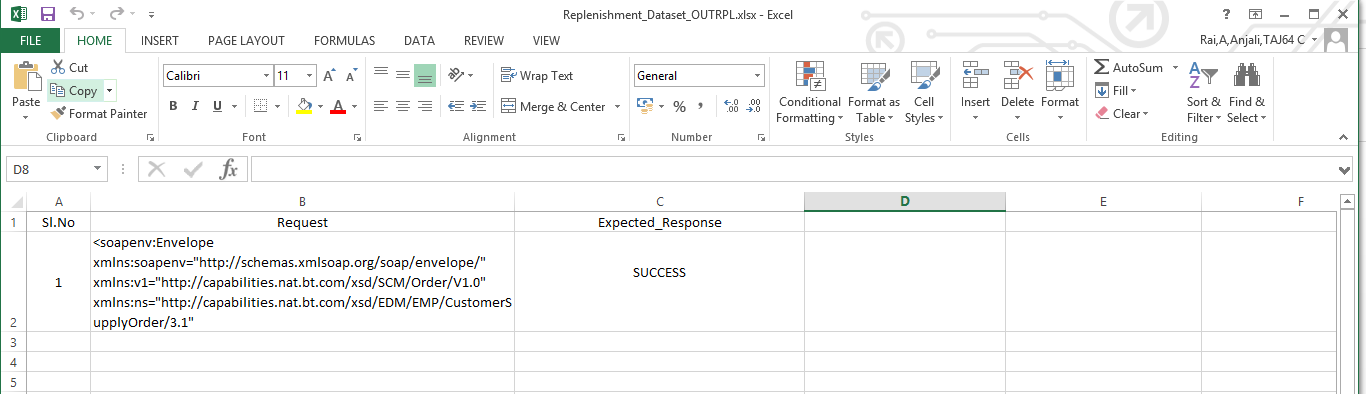
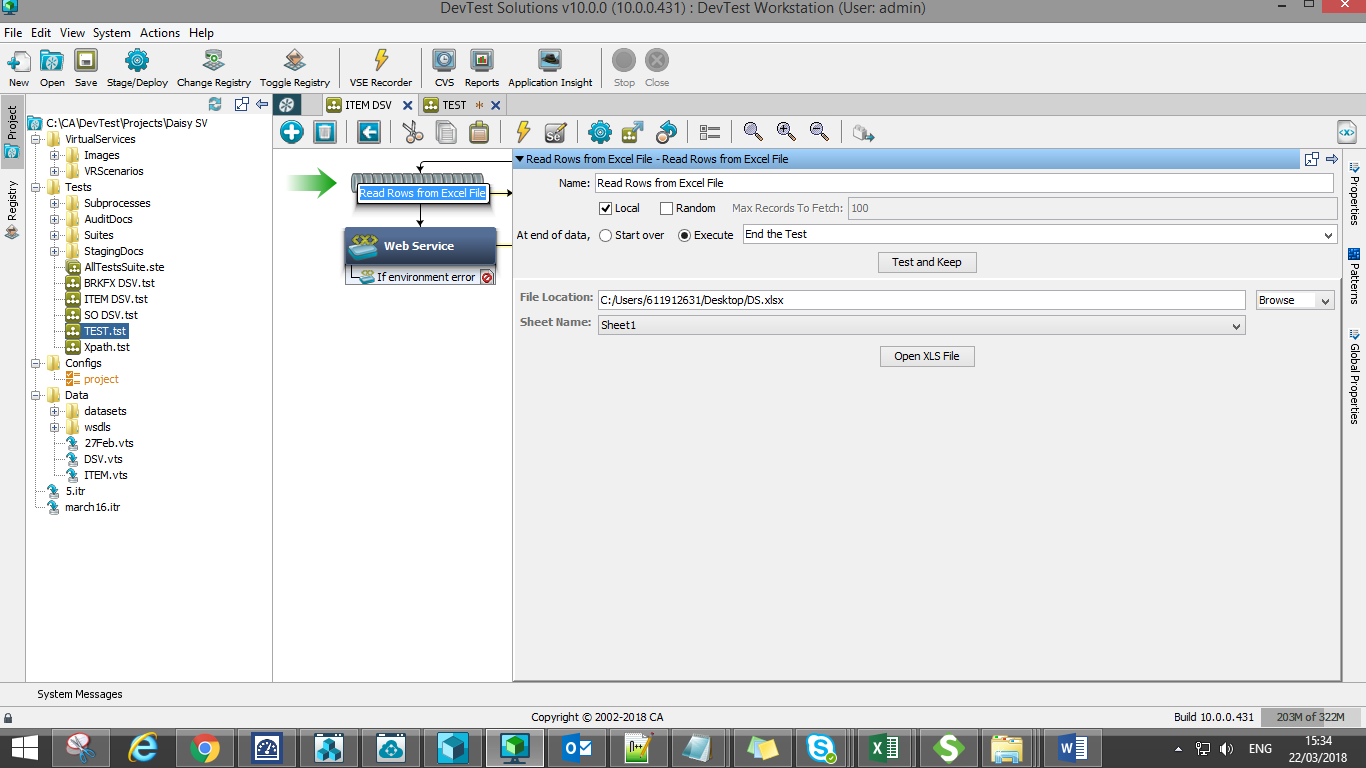
# Steps to generate request XML from SV

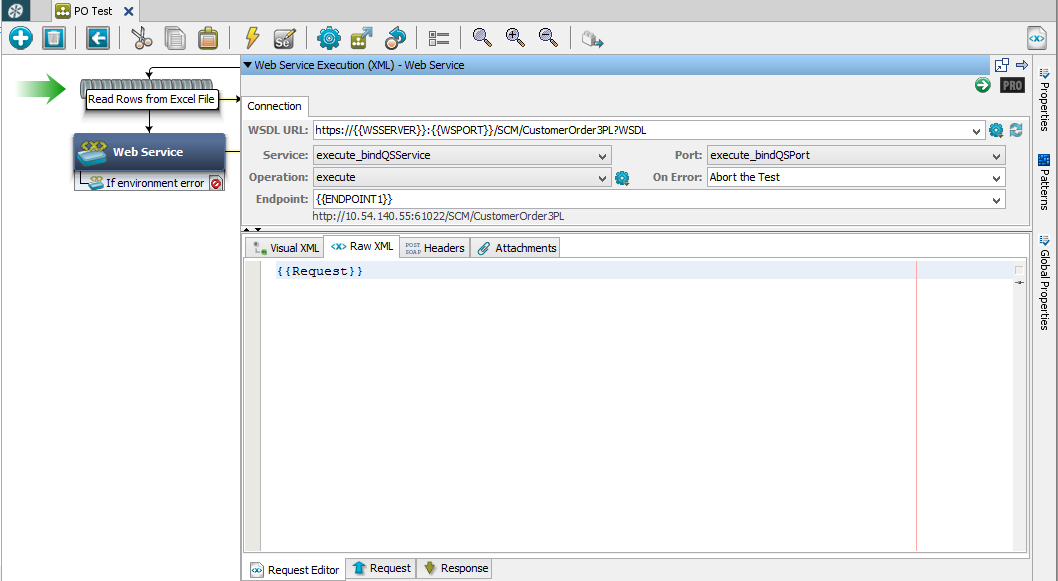
1. Open DevTest create a new project
2. Create a New Test case and name it
3. Click on add button to add new **Web Service Execution(XML)** step from **Web/Web Services** drop-down list



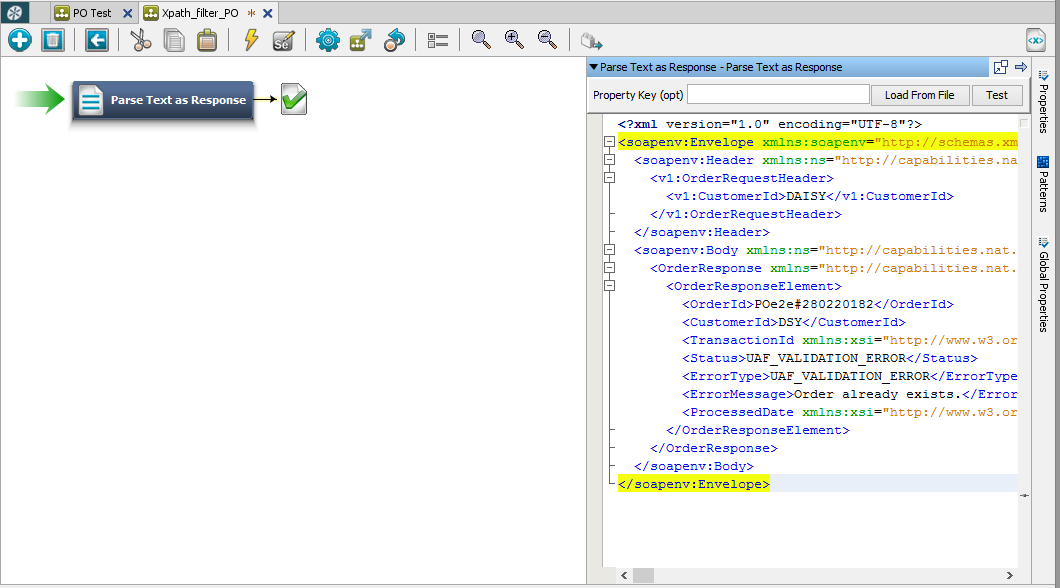
1. Select WSDL file (file from PAL team) in **WSDL URL** field from where it is saved
2. **WSDL details** will be populated automatically in Web Service Execution (XML) step, like below

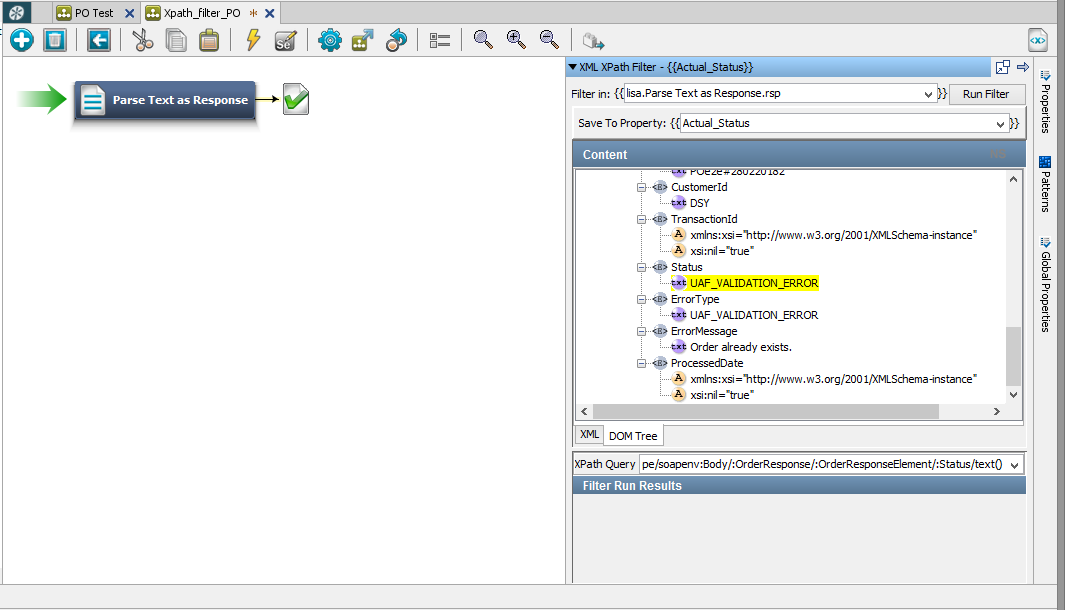


1. Mean While, prepare Excel sheet with set of required request that need to be fired to PAL in the absence of DAISY from SV, also add C column, add Expected response to validate the response 
2. Add **Read Rows from Excel File** Data set and select the data sheet (excel spread sheet that we prepared in last step)
3. Browse the file location where the excel sheet is saved. To view the sheet in DevTest click on open XLS file
4. Click onto the below **Raw XML** tab to load the request. In this case as excel dataset is used, specify the column name under which request data is present in **parenthesis**

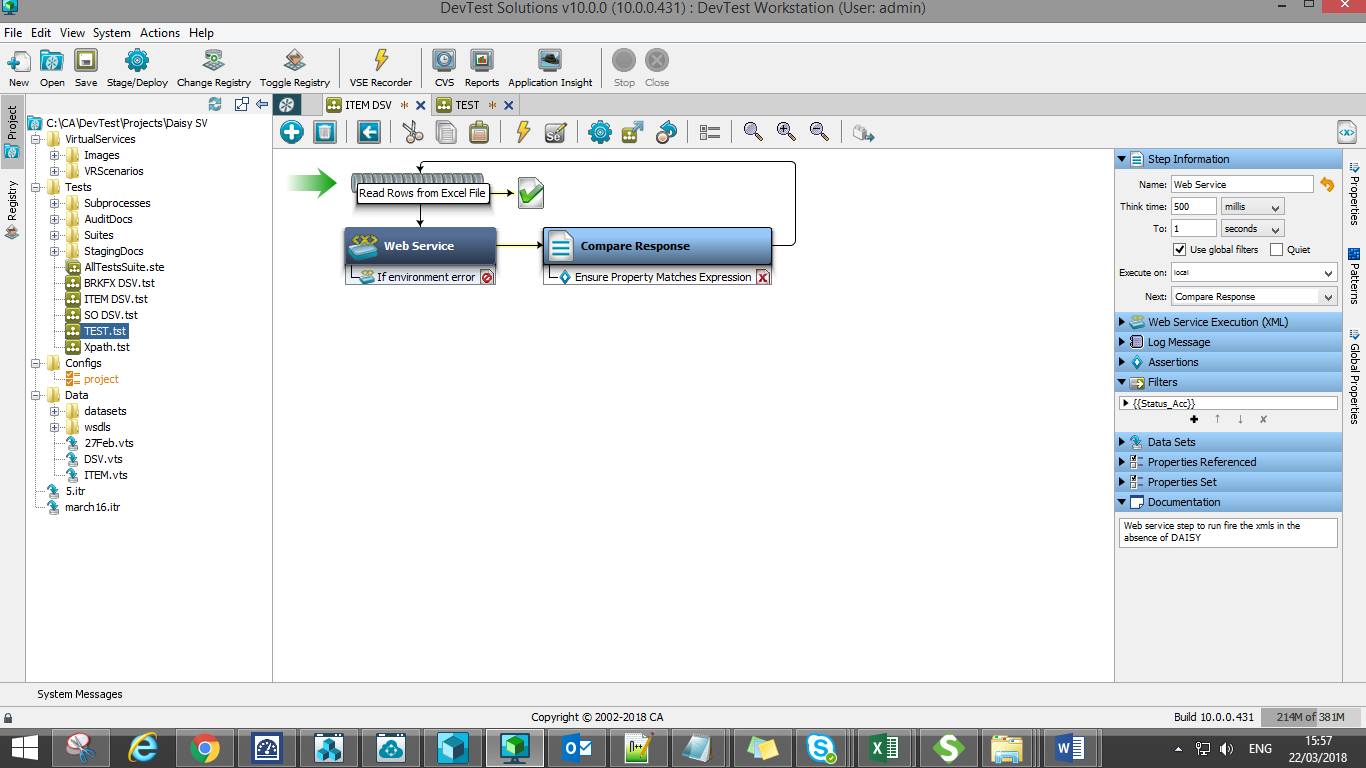


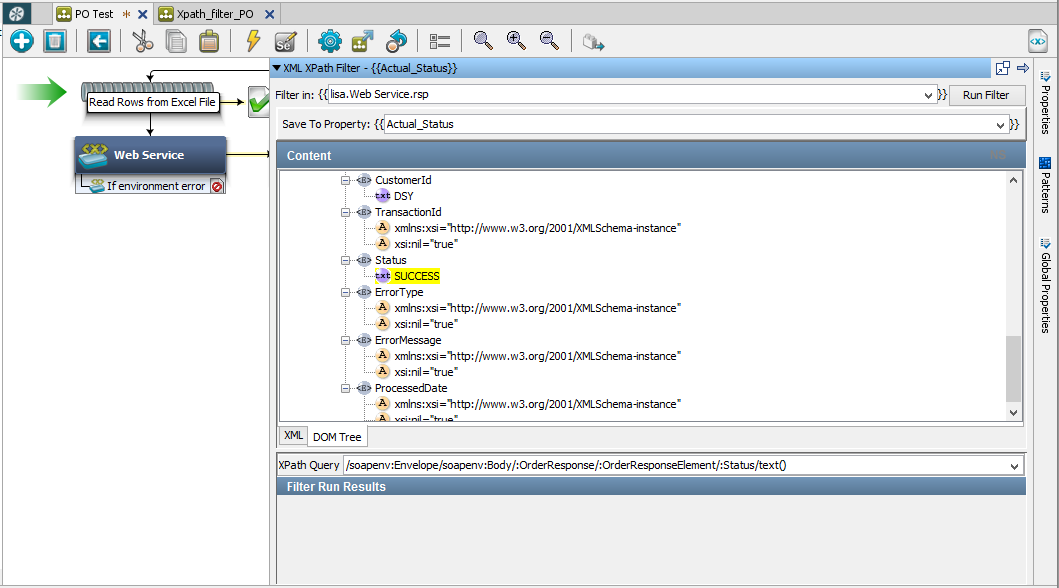
# To Capture Response

1. Create a new test case add **Parse Text as a Response step.** Copy the responses.
2. Add **XPath filter** to the specific field required and save the property name. In this case filter is applied on status field

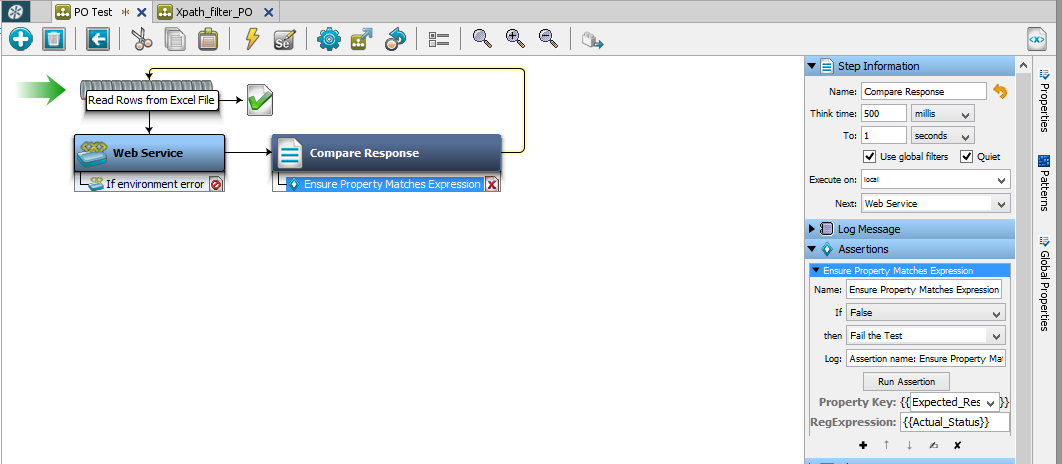


1. Go back to the test case where request files are loaded and copy the XPath filter to the Web Service step

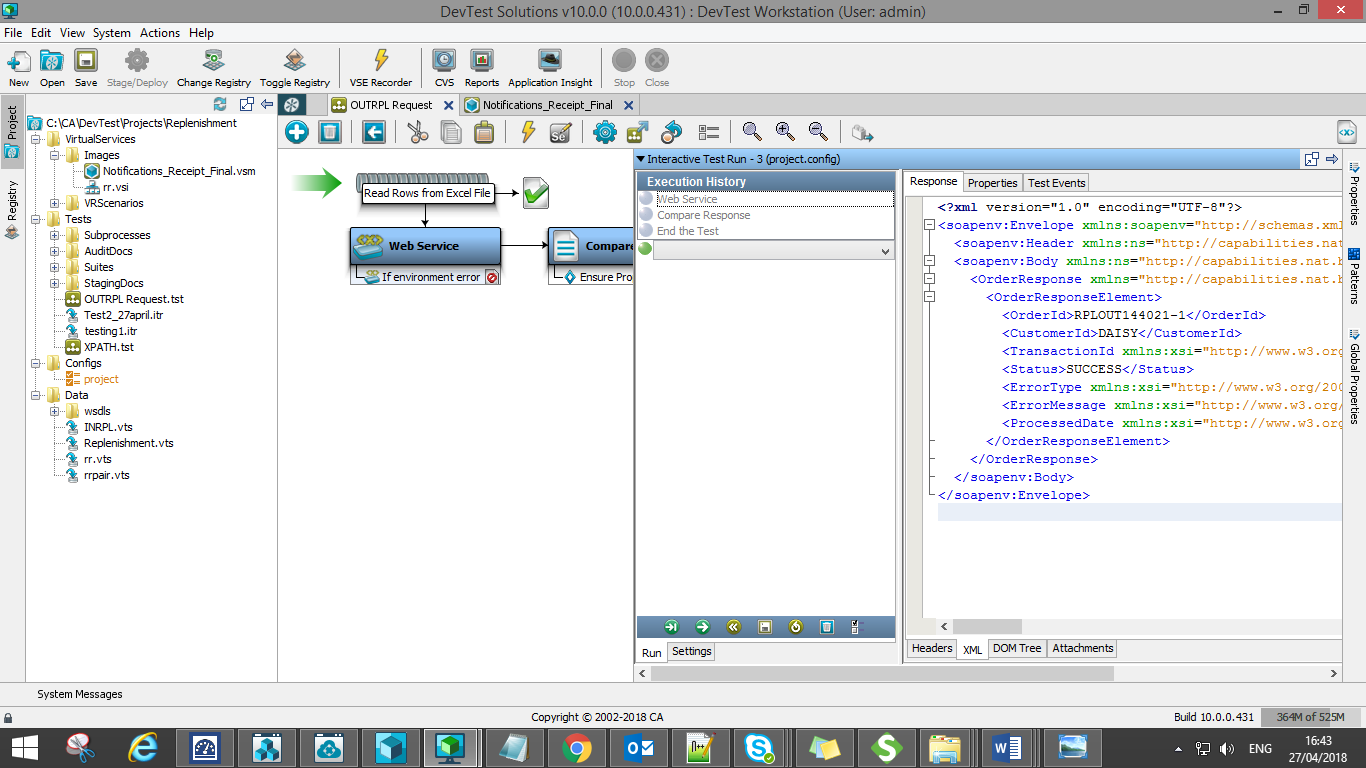




1. Add **Do-Nothing Step** next to Web service step. Add **Ensure Property Matches Expression assertion** in the following step. In the assertion give **Property Key field value** same as the **column name of excel sheet** in which the response is present and **Regular Expression field value** same as the **property name used in XPath filter**



1. Save and run the test case

****

1. **Test Result:**

**PASS**:

If there is no mismatch i.e., **Expected** and **Actual responses** are same, ITR will successfully run till end of the loop (end of test data in spread sheet) and ends the test.

**FAIL**:

In case of mismatch in **Expected response** we entered in data set spread sheet and the **Actual response** we get from SC UAF system, test case (ITR) will fail and ends the test irrespective of test data present in data set spreadsheet.

**Corrective Action:** Validate the reason for the error and rerun the test.

**Sample Documents:**



**Data Set spreadsheet:**



**SV References:**



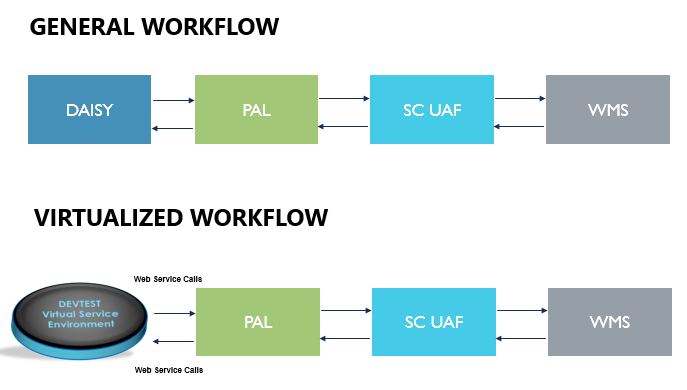
# Replenishment Process

**Plan:**

To virtualize replenishment process in the absence of DAISY, from SV. First create a web service TEST CASE using the wsdl details shared by testing team and add step to read data from data sheet and also add Do Nothing step to validate synchronous response received by SV. As soon as the Order is received BT component, it has to be processed in BT component and SV receives corresponding notification for the processed status. Once shipped notification is received, SV should automatically trigger INRPL request and wait to validate receipt notification for all shipped quantity. Thus the replenishment process is accomplished by using SV tool DevTest.

1. Request: VS (OUTRPL XML) > SC UAF > WMS
2. Response: SC UAF > VS (synchronous response)
3. Status Notification: PAL > VS (released, picked & shipped)
4. Request: VS (INRPL XML) > SC UAF > WMS
5. Response: SC UAF > VS (synchronous response)
6. Receipt Notification: PAL > VS (till receipted completely)

**Workflow:**



**Replenishment Process Flow:**

1. Daisy will send an OUTRPL xml type order to BT
2. BT will respond to the synchronous request saying success to Daisy
3. BT will provide a release notification to Daisy
4. BT will provide a pick notification to Daisy
5. BT will provide a ship notification to Daisy
6. Daisy will use the ship notification to create the INRPL xml type order for BT – there is a co-relation between some order numbers of OUTRPL and INRPL
7. BT will respond to the synchronous request saying success to Daisy
8. BT will provide a or multiple receipt responses to Daisy

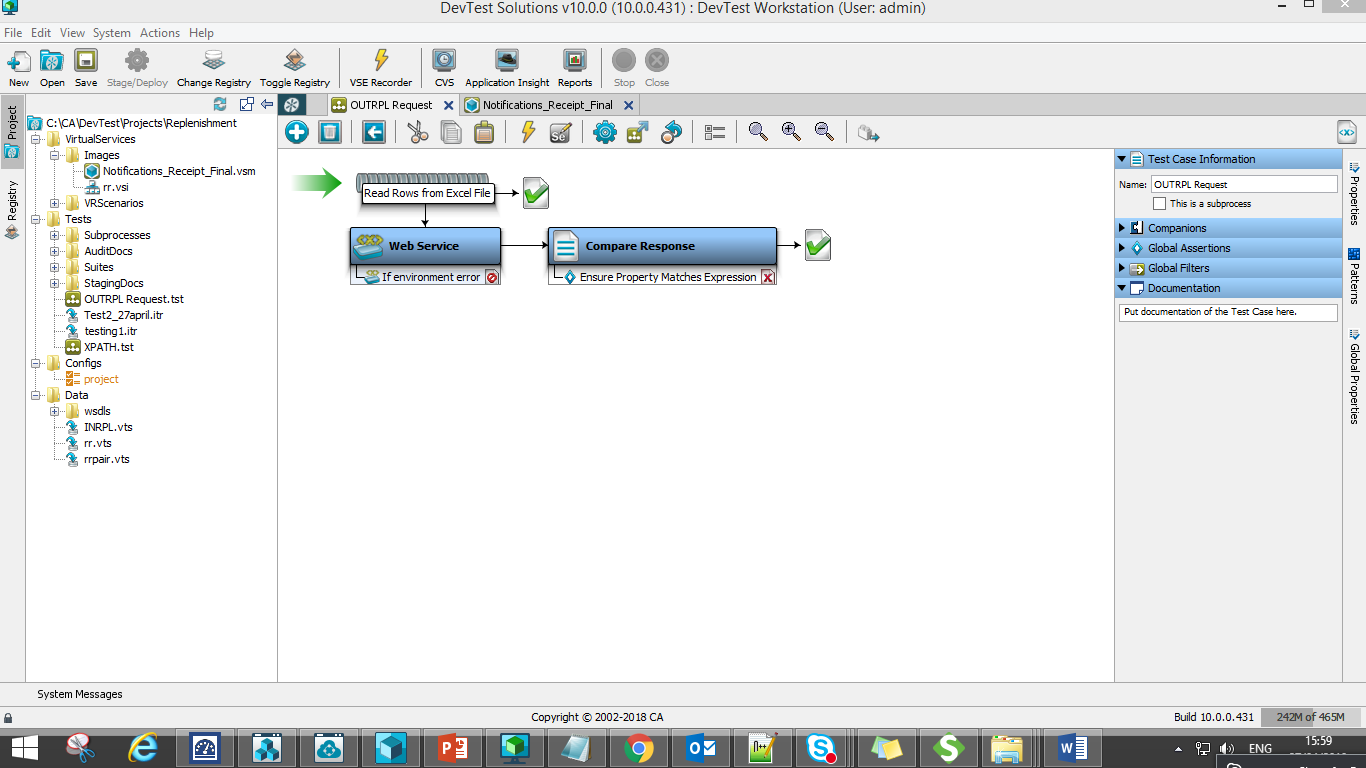
**Use Case:**

1. Ability to create an OUTRPL order and invoke the PAL webservice to push the xml to BT
2. Validate the response to this synchronous response
3. Validate the release, pick, and ship notifications are received successfully with all required attributes
4. Use the ship notification as a trigger to create an INRPL order and invoke the PAL web services to push the xml to BT
5. Validate the response to this synchronous response
6. Validate the receipt response from BT against the IN RPL order – the service should accept receipt responses as long as the item QTY sent in the INRPL order has been fully receipted
   1. In the items quantity to be receipted is 10 – BT may send 1 receipt of 10 QTY, or 10 receipts of 1 QTY each, or any such combinations of multiple receipt
   2. However BT should not send receipt where qty sums up to be > 10 – this should report an error

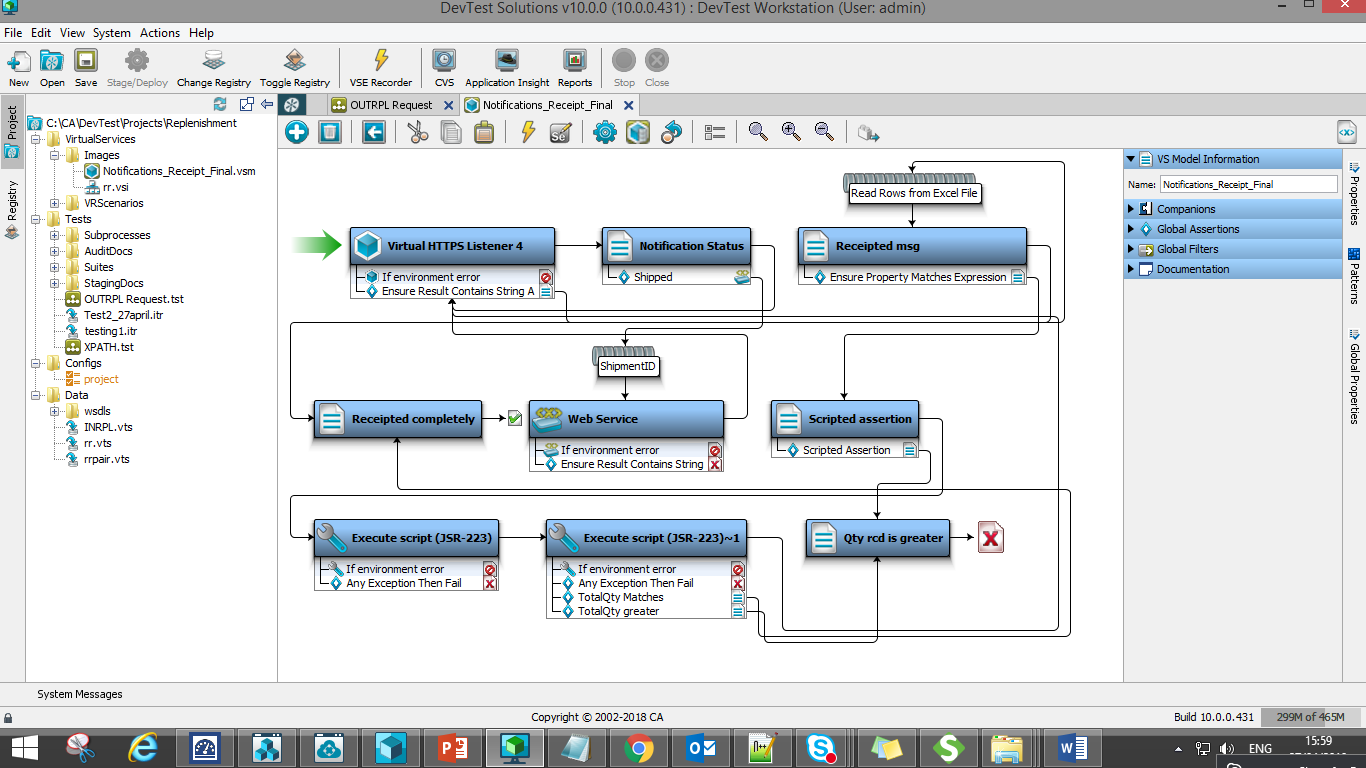
# Validating OUTRPL notification and automating INRPL request

1. **Step 1:** Firing XML (OUTRPL) from SV (DAISY) to SC UAF system and verifying the response received (synchronous message from SC UAF for the fired xml). Refer [“Steps to generate request XML from SV”](#_Steps_to_generate) for more details.

Note: Noted down the OriginationSystemOrderRef id from OUTRPL which will be referred as CustomerPoNo in INRPL

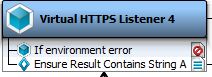


1. **Step 2: VSM:**

****

**A Listener step** is added to listen to the incoming notification from PAL component for the OUTRPL xml order processed. Five **XML XPath filters** were added to capture property values and one **assertion** was also added to the listener step.

**Assertion: To validate whether the notification is “Order Status Notification” or “Receipted Notification”**





Five **XML XPath filters** to capture the following values from the notification:

* **Notification1:** Order Status notification of Released or Picked or Shipped notification xml

**XPath Query:** **/soapenv:Envelope/soapenv:Body/osu:OrderStatusUpdateRequest/osu:OrderStatusUpdateElement/osu:OrderStatus/text()**

* **OriginationSystemOrderRef**: <<Which will be referred as Customer PO No. in INRPL)

**XPath Query: /soapenv:Envelope/soapenv:Body/osu:OrderStatusUpdateRequest/osu:OrderStatusUpdateElement/osu:OriginationSystemOrderRef/text()**

* **CustomerItemNumber:** Item No should be same in both OUTRPL & INRPL request

**XPath Query:**

**/soapenv:Envelope/soapenv:Body/osu:OrderStatusUpdateRequest/osu:OrderStatusUpdateElement/p1:OrderShipmentDetail[1]/p1:CustomerItemNumber/text()**

* **Qty\_Rcd:** count(xml XPath): gives us the count of no. of section present

**XPath Query: count(/soapenv:Envelope/soapenv:Body/p:WarehouseReceiptRequest/p:WarehouseReceiptElement)**

**Here: count is 5**

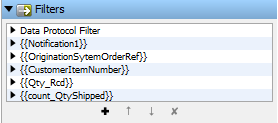
****

* **Count\_QtyShipped:** count(xml xpath): gives us the count of no.of section present

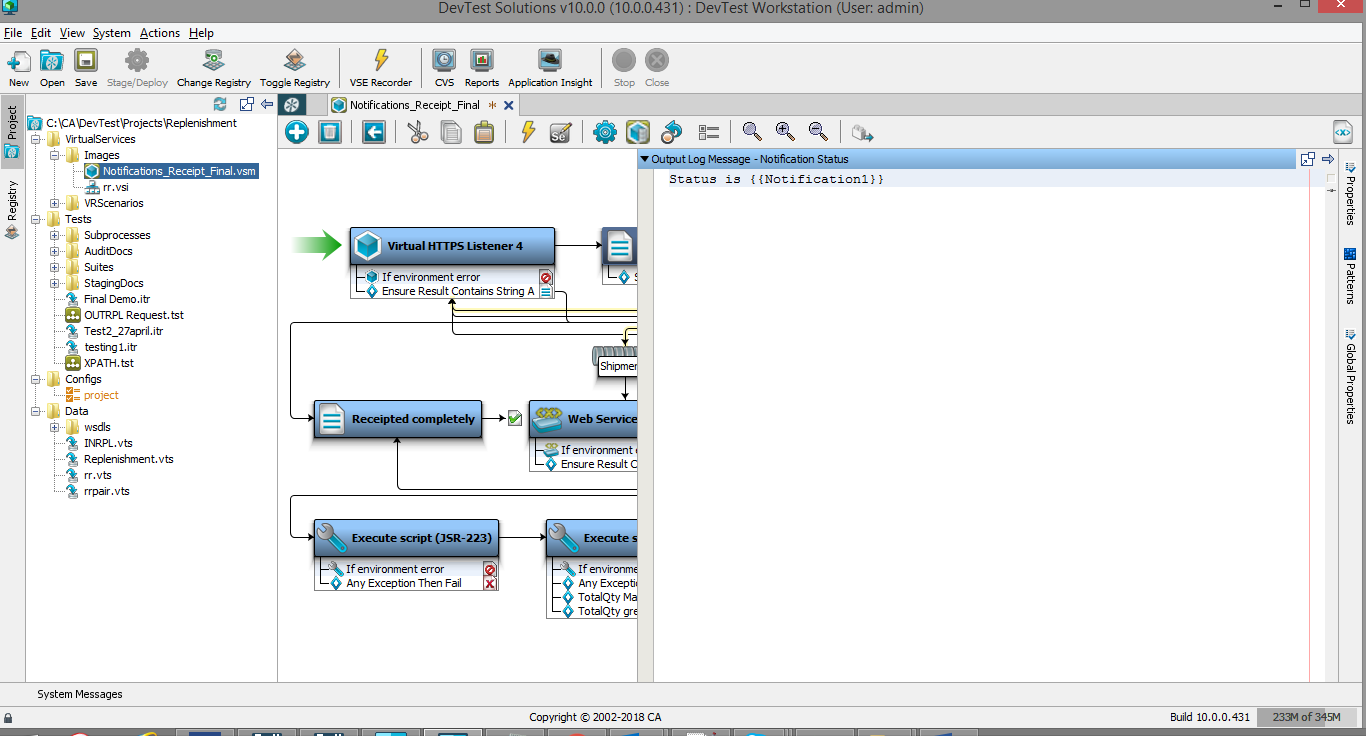
**XPath Query: count(/soapenv:Envelope/soapenv:Body/osu:OrderStatusUpdateRequest/osu:OrderStatusUpdateElement/p1:OrderShipmentDetail)**

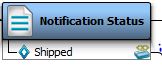
**Here: count is 5**

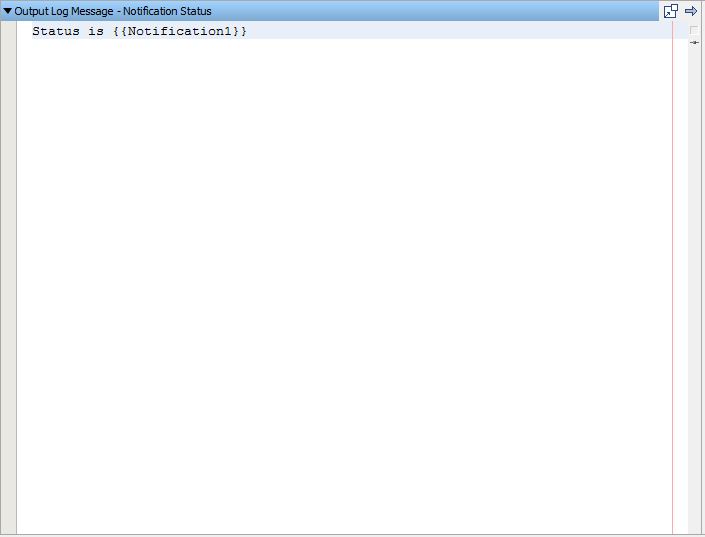




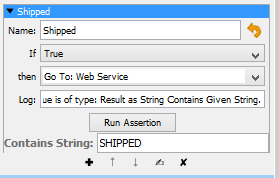
1. When the listener step listens to the incoming notification it displays the notification in the **output log message** step
2. Here **Notification1** corresponds to the property name which is used to capture notification status in the **listener step**

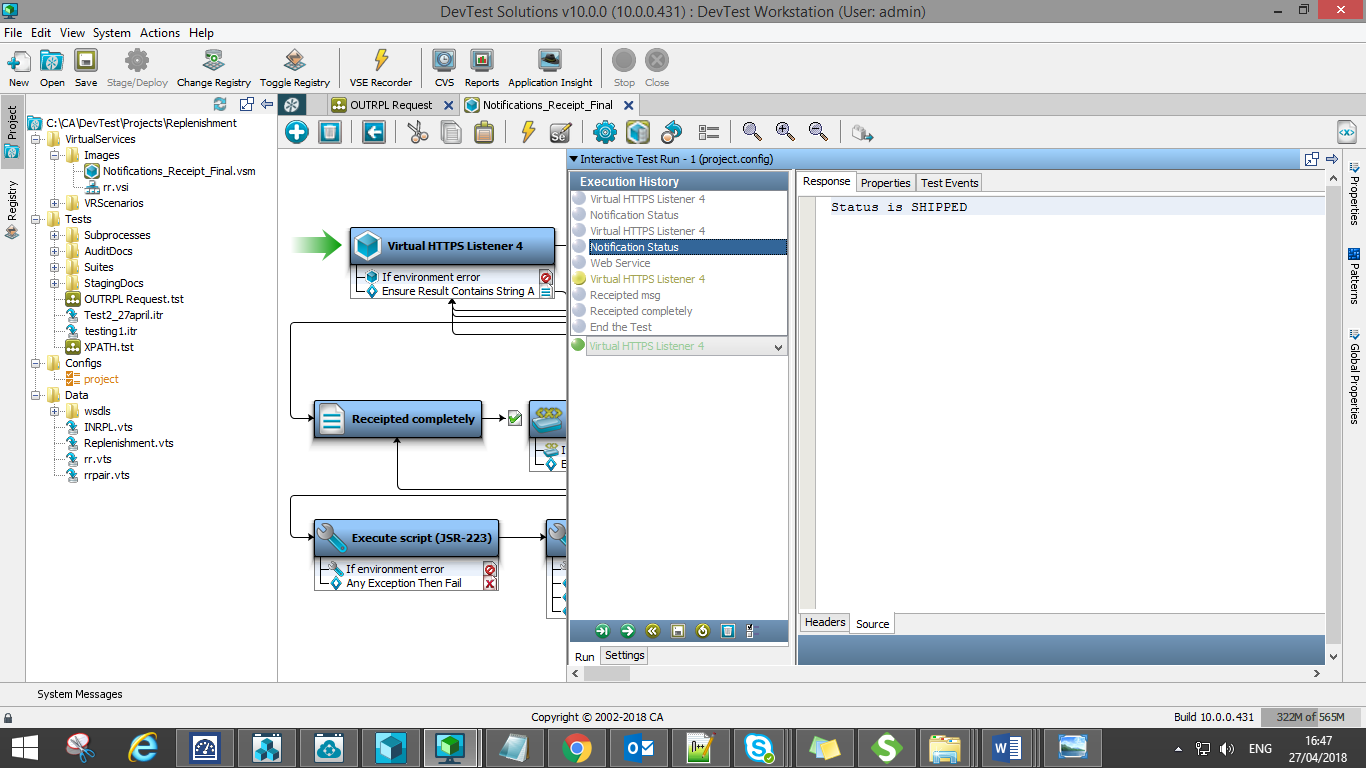






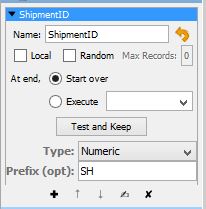
1. **Ensure Result contain String Assertion** is added to this step to look for **Shipped** status notification.

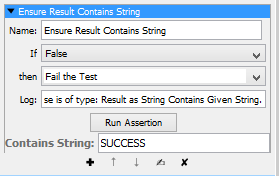


****

1. Listener step will listen to N number of request that comes in till it get the **Shipped** notification based on the **Ensure Result contain String Assertion** added here.
2. Once the shipped notification is received and displayed in **output log message** it automates INRPL request to trigger in **web service** step
3. This fires xml an INRPL request to SC UAF system
4. This step contains **one assertion** to validate synchronous response when INRPL request is triggered. **ShipmentID** is **Unicode Generator step** which generates **unique order Id** for each INRPL order.

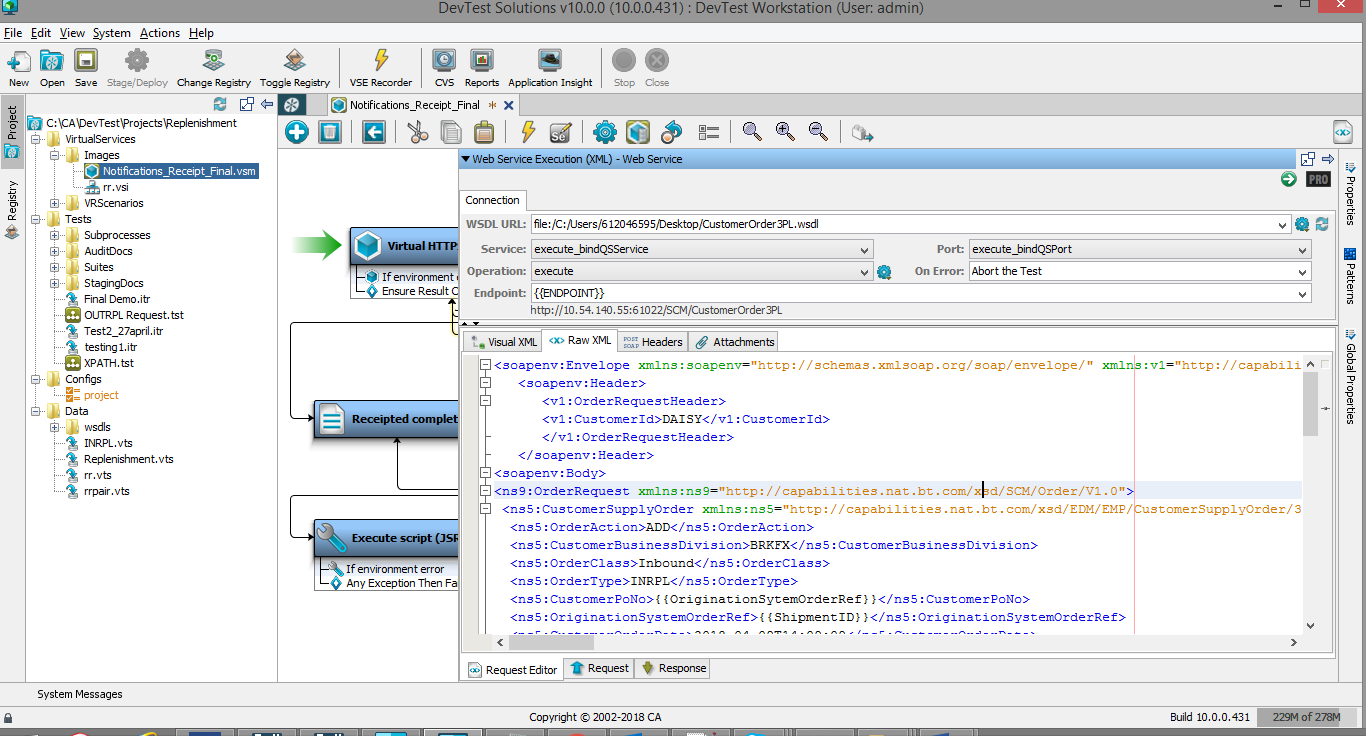


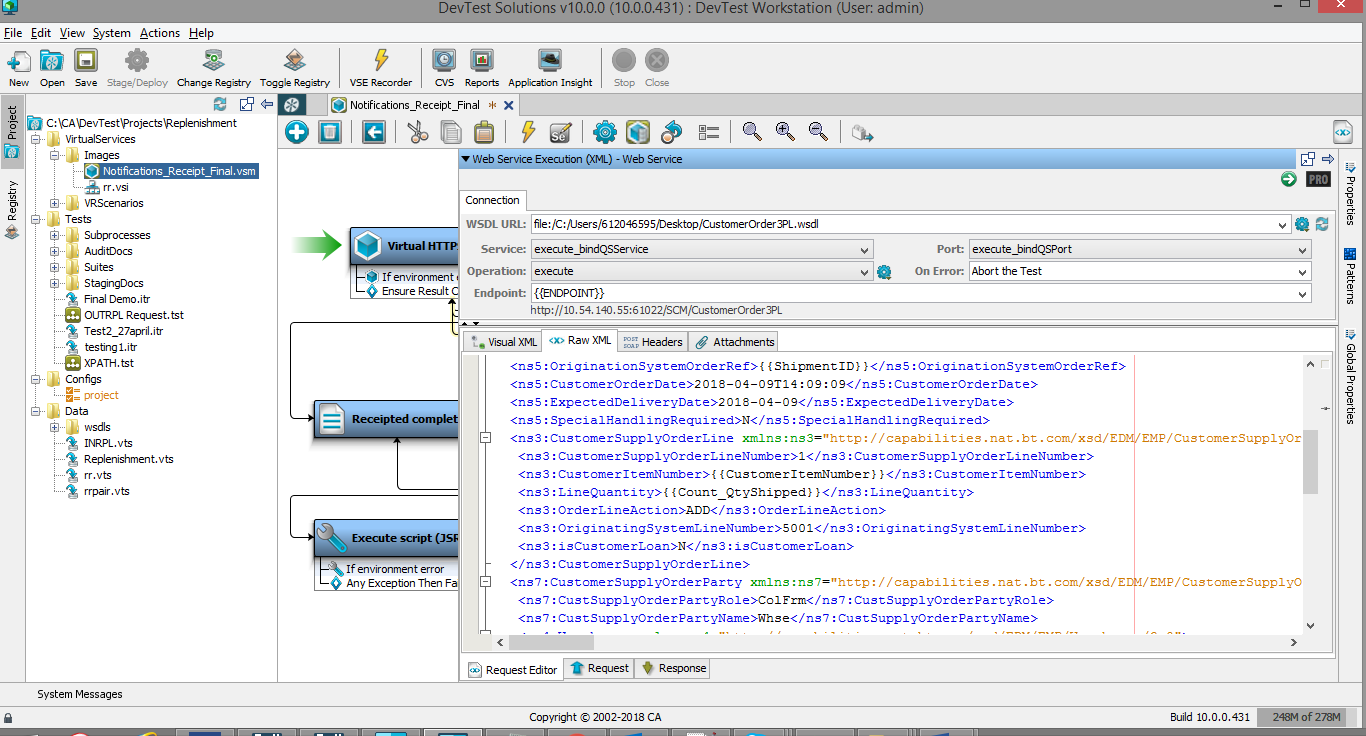


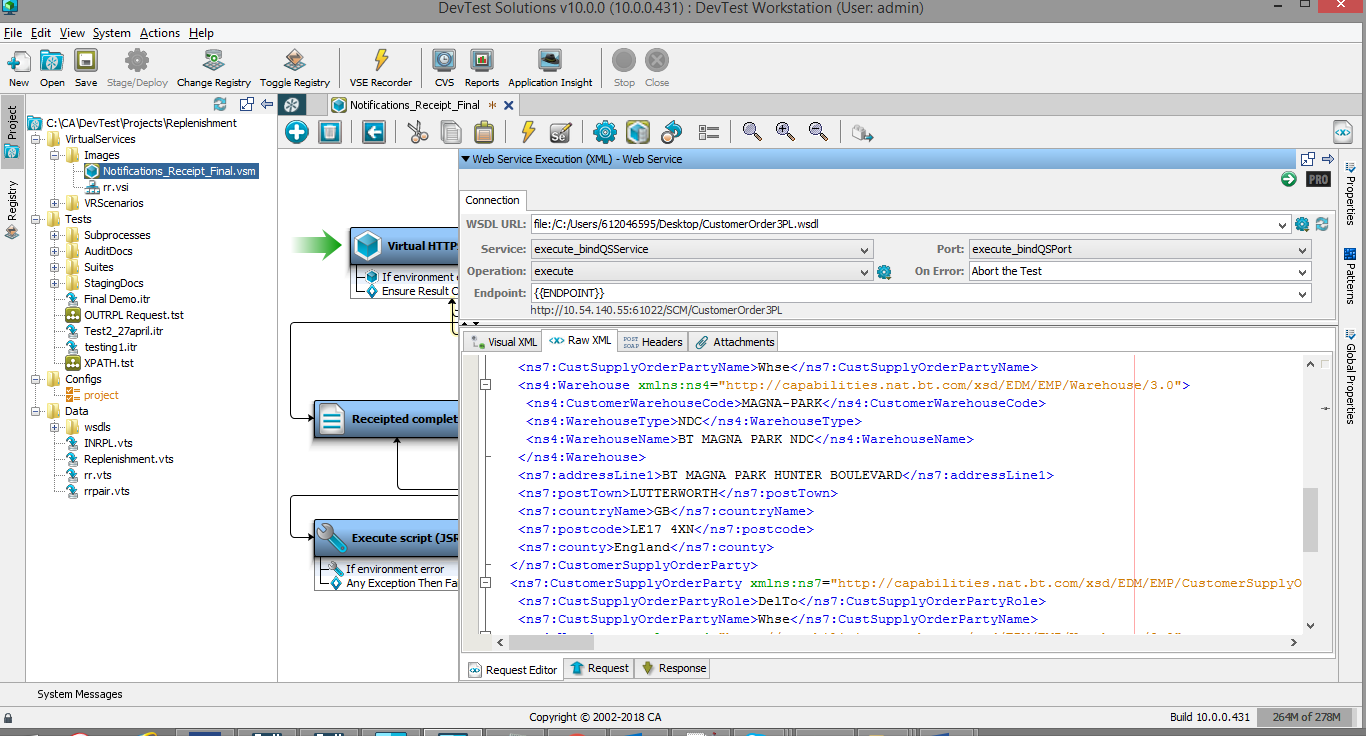


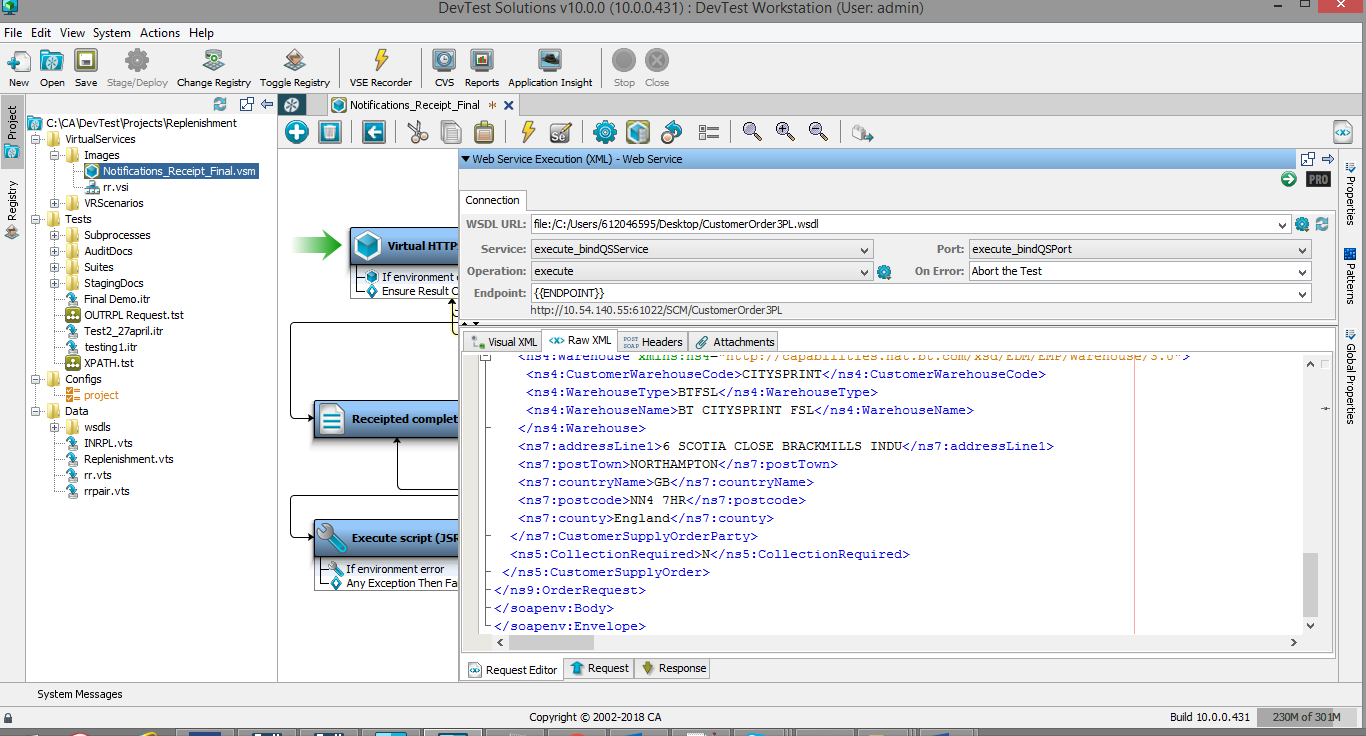


**INRPL request in We Service step with properties referred:**

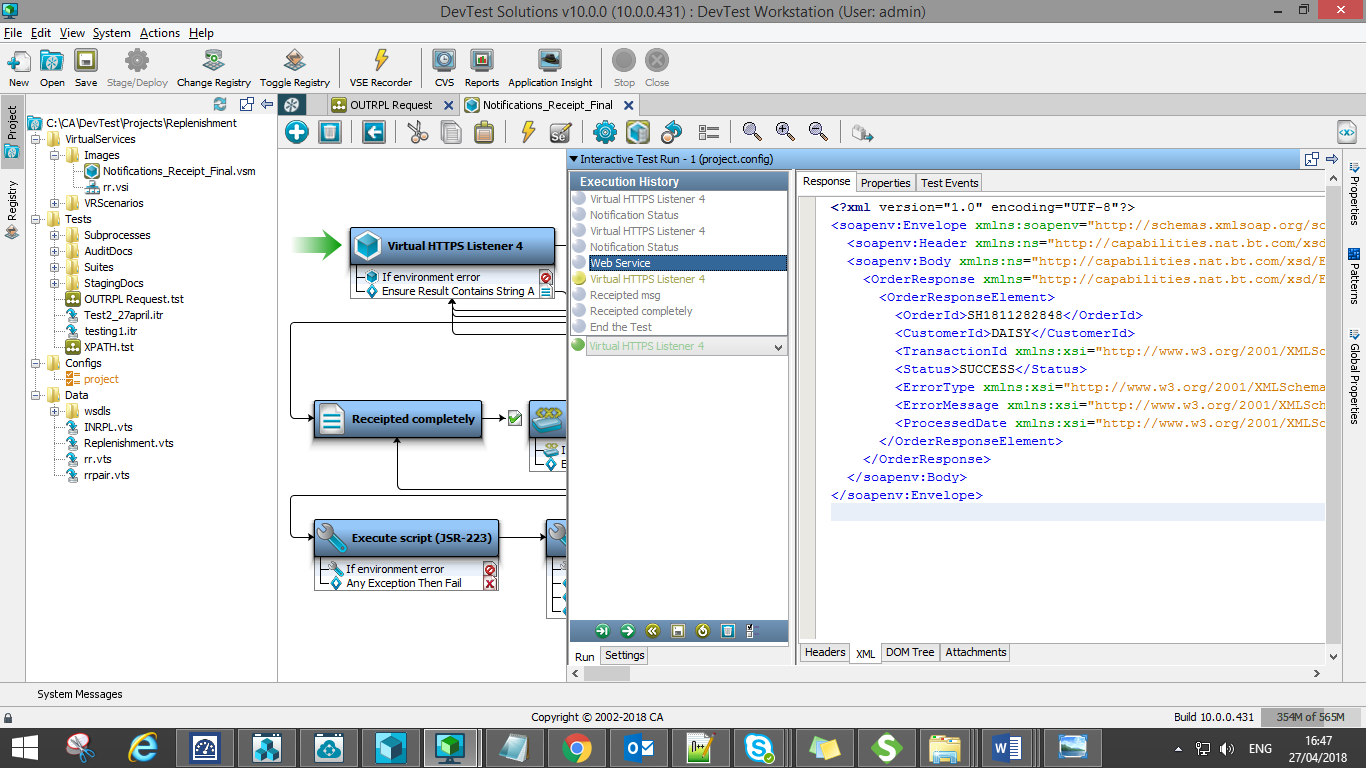






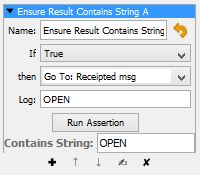


**Synchronous message for fired INRPL request:**

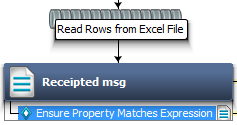
****

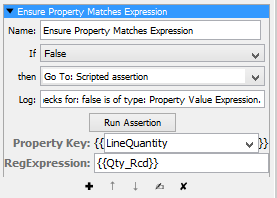
1. After verifying synchronous response, (same to OUTRPL response verification) it redirects back to **listener step** to listen to the **receipted notification**. The listener step waits to receive asynchronous receipt notification for shipped quantity with String “OPEN”.
2. Sample Receipted notification:



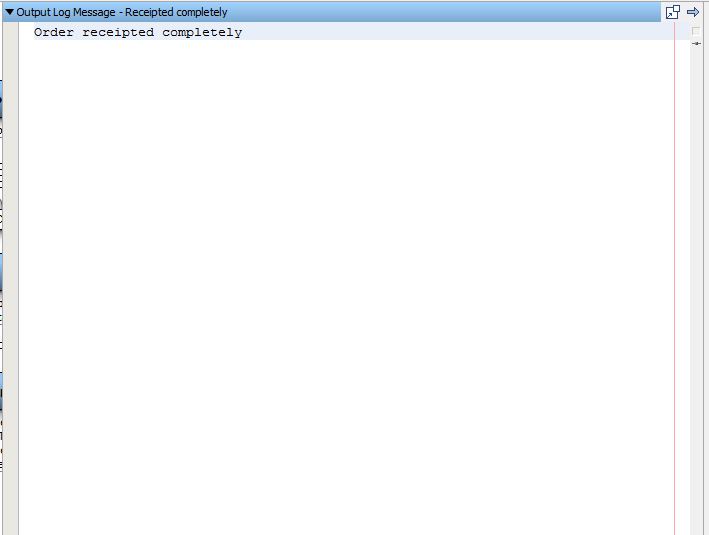


1. When listener step encounters **OPEN** status it directs to **Do-Nothing-Step** to compare **quantity received** value with **line quantity** value of INRPL request using Ensure Property Matches Expression Filter.
2. This **line quantity** value flows from INRPL request what we are firing for how much quantity. Based on that open data sheet and change the value accordingly.

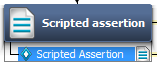


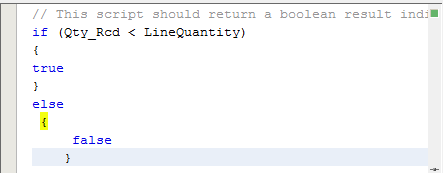


1. If the **quantity received** value is **equal** to the **line quantity** of INRPL request then it proceeds to **Output log message** step displaying Order receipted completely and end the test.

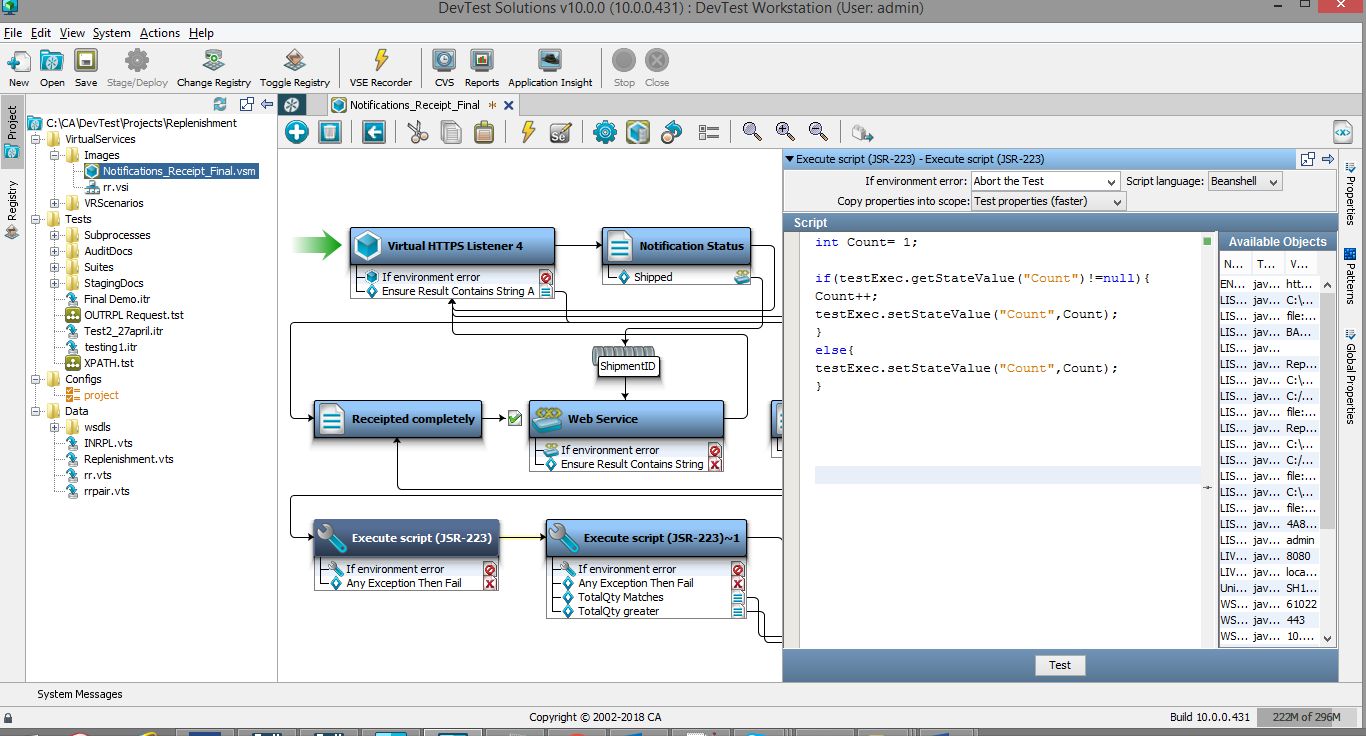


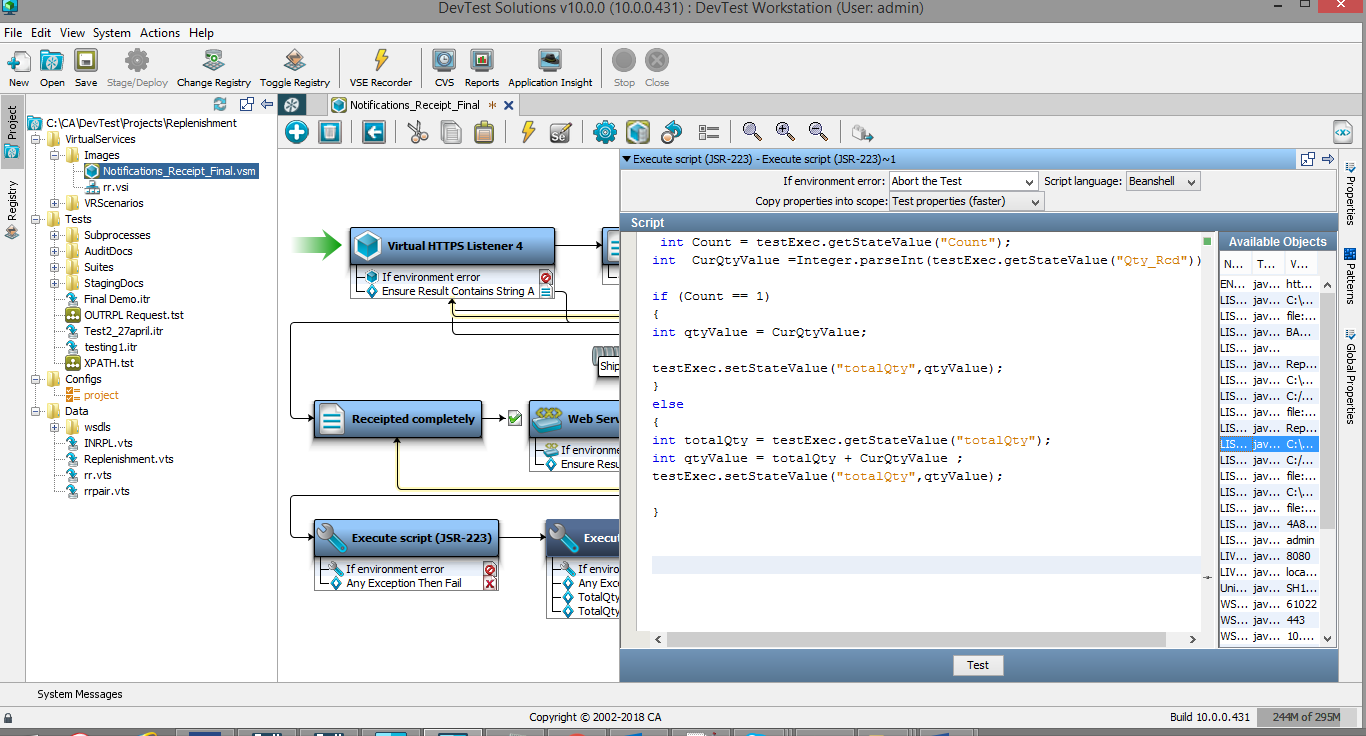
1. If both values are **not matching** then it directs to **Do-Nothing-Step** where it checks whether the **quantity received** is **less than line quantity** of INRPL request using scripted assertion.

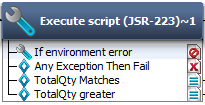


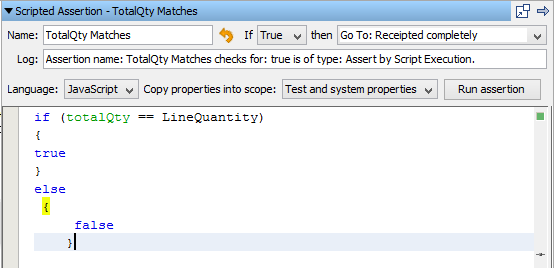


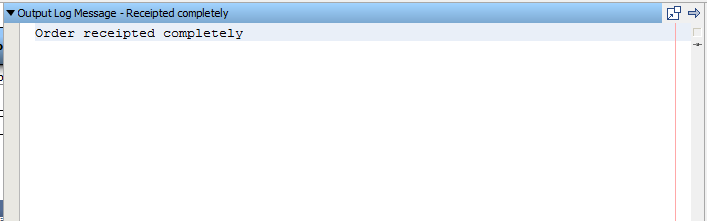
1. If the scripted assertion is true it proceeds to next step i.e **Java Execution step** where it **executes** JavaScript code to **store** the **quantity received value** and **sum up**. JavaScript step contains **two** scripted assertions, **to check sum of total quantity is equal to line quantity** and **sum of total quantity is greater than line** **quantity**. Until the sum of quantity received is equal to line quantity it redirects to listener step to listen to the remaining quantity to be received. Once the sum of quantity received is equal to line quantity it proceeds to **output log message** displaying **Order receipted completely** and end the test
2. **JAVA script** steps to count the value of quantity received every time whenever the receipted notification is received:



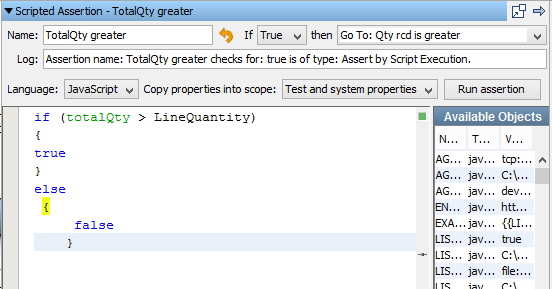


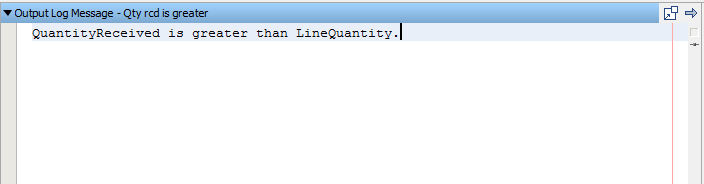




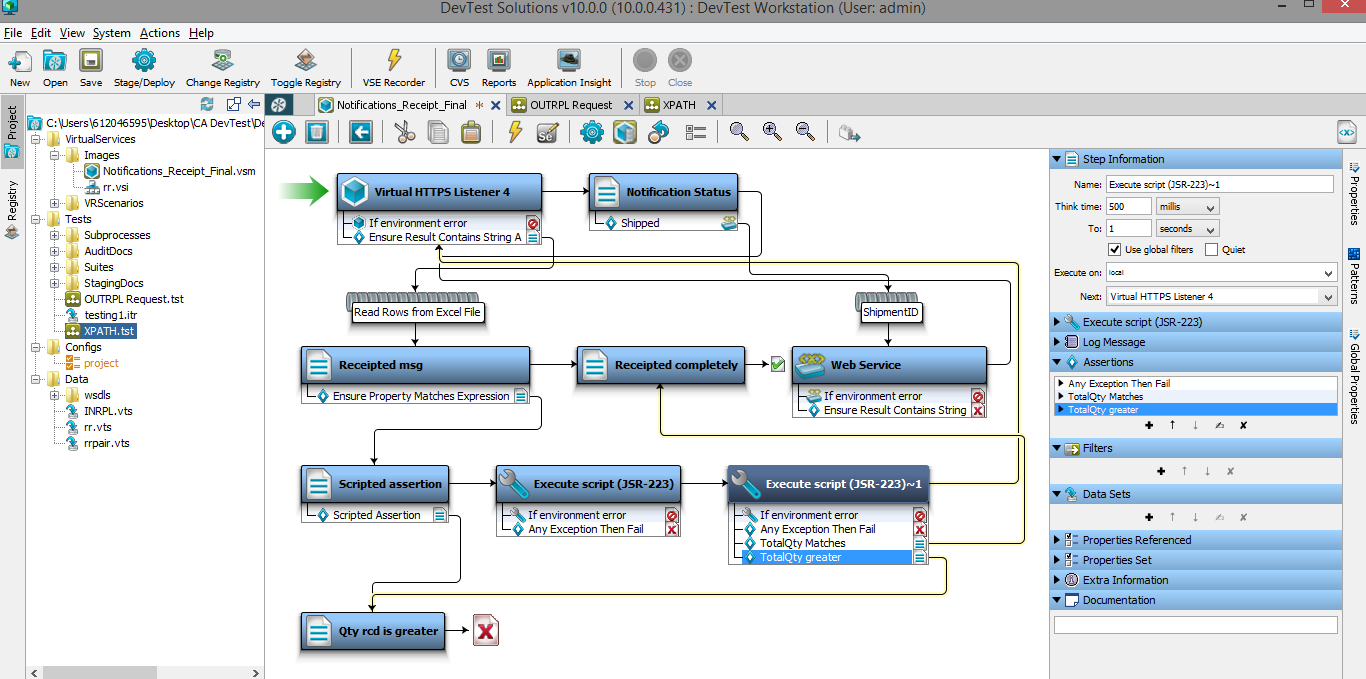


1. If the above scripted assertion is false or the sum of quantity received is greater than line quantity then it proceeds to output log message displaying **Quantity Received is greater than Line Quantity** and fail the test.





**The complete VSM:**



**References:**

