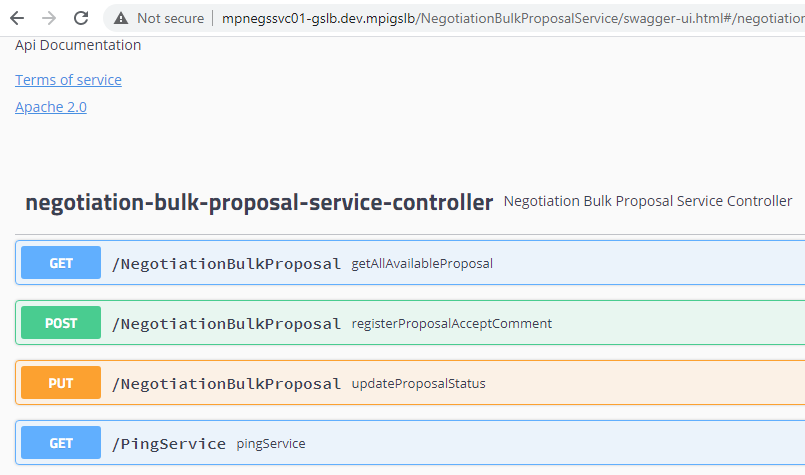
# Learning

## Mechanism to Invoke REST Service

### Creating REST Business Service

List of External JAX\_RS Operations:-



1. Operation Name:-getAllAvailableProposal

**http://mpnegssvc01-gslb.dev.mpigslb/NegotiationBulkProposalService/NegotiationBulkProposal?proposalId=1234567788**

1. Operation Name:-registerProposalAcceptComment

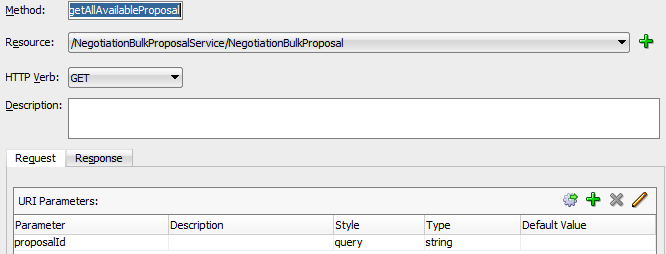
**http://mpnegssvc01-gslb.dev.mpigslb/NegotiationBulkProposalService/NegotiationBulkProposal**

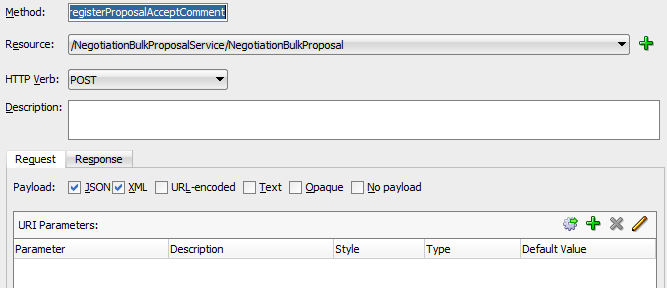
1. Operation Name:-updateProposalStatus

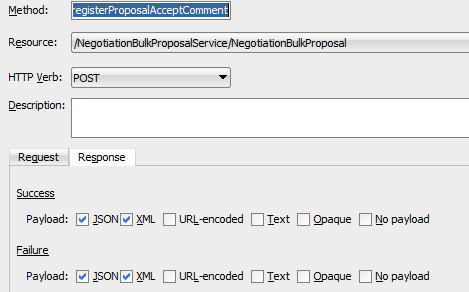
**http://mpnegssvc01-gslb.dev.mpigslb/NegotiationBulkProposalService/NegotiationBulkProposal**

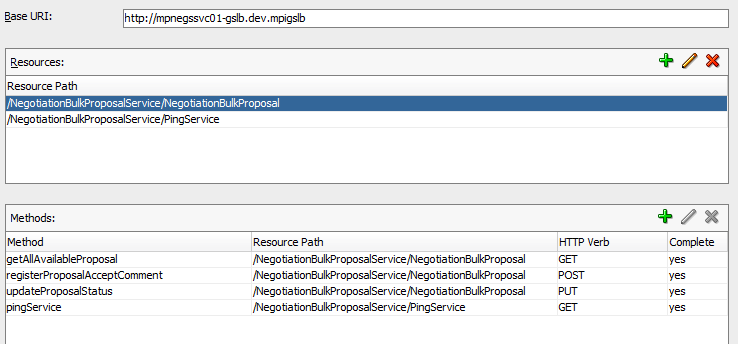
1. Operation Name:-pingService

**http://mpnegssvc01-gslb.dev.mpigslb/NegotiationBulkProposalService/PingService**









After configuration it will create WADL file:-

NegotiationBulkProposalServiceBS.WADL:-

<?xml version = '1.0' encoding = 'UTF-8'?>

<application xmlns:soa="http://www.oracle.com/soa/rest" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://wadl.dev.java.net/2009/02">

<doc title="NegotiationBulkProposalServiceBS">RestReference</doc>

<resources>

<resource path="/NegotiationBulkProposalService/NegotiationBulkProposal">

<method name="GET" soa:name="getAllAvailableProposal">

<request>

<param name="proposalId" style="query" type="xsd:string"/>

</request>

<response status="200">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

<response status="">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

</method>

<method name="POST" soa:name="registerProposalAcceptComment">

<request>

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</request>

<response status="200">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

<response status="">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

</method>

<method name="PUT" soa:name="updateProposalStatus">

<request>

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</request>

<response status="200">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

<response status="">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

</method>

</resource>

<resource path="/NegotiationBulkProposalService/PingService">

<method name="GET" soa:name="pingService">

<request>

<param name="parameter" style="query" default="text" type="xsd:string"/>

</request>

<response status="200">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

<response status="">

<representation mediaType="application/json"/>

<representation mediaType="application/xml"/>

</response>

</method>

</resource>

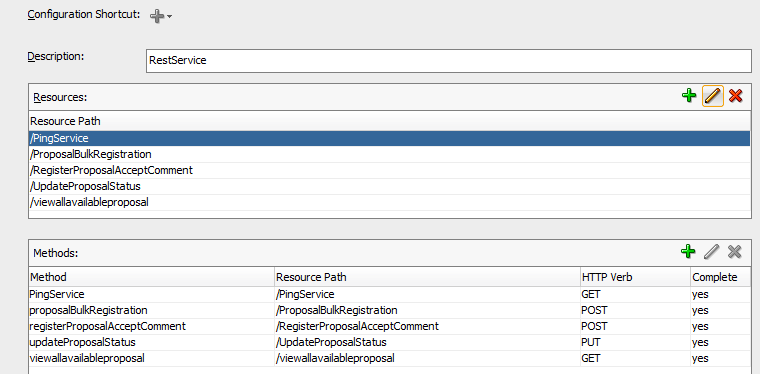
</resources>

</application>

### Creating REST Proxy Service

Drag and drop REST Adapter from Component Panel to Proxy Services Swim Line.

Here we can give any Resource Path as per our will. It doesn’t matter what is BS resource URI.



ProposalBulkRegistrationRST.WADL:-

<?xml version = '1.0' encoding = 'UTF-8'?>

<application xmlns:soa="http://www.oracle.com/soa/rest" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://wadl.dev.java.net/2009/02">

<doc title="ProposalBulkRegistrationRST">RestService</doc>

<resources>

<resource path="/ProposalBulkRegistration">

<method name="POST" soa:name="proposalBulkRegistration">

<request>

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</request>

<response status="200">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

<response status="500">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

</method>

</resource>

<resource path="/viewallavailableproposal">

<method name="GET" soa:name="viewallavailableproposal">

<request>

<param name="proposalId" style="query" type="xsd:string"/>

</request>

<response status="200">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

<response status="500">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

</method>

</resource>

<resource path="/RegisterProposalAcceptComment">

<method name="POST" soa:name="registerProposalAcceptComment">

<request>

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</request>

<response status="200">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

<response status="500">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

</method>

</resource>

<resource path="/UpdateProposalStatus">

<method name="PUT" soa:name="updateProposalStatus">

<request>

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</request>

<response status="200">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

<response status="500">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

</method>

</resource>

<resource path="/PingService">

<method name="GET" soa:name="PingService">

<request>

<param name="parameter" style="query" type="xsd:string"/>

</request>

<response status="200">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

<response status="500">

<representation mediaType="application/xml"/>

<representation mediaType="application/json"/>

</response>

</method>

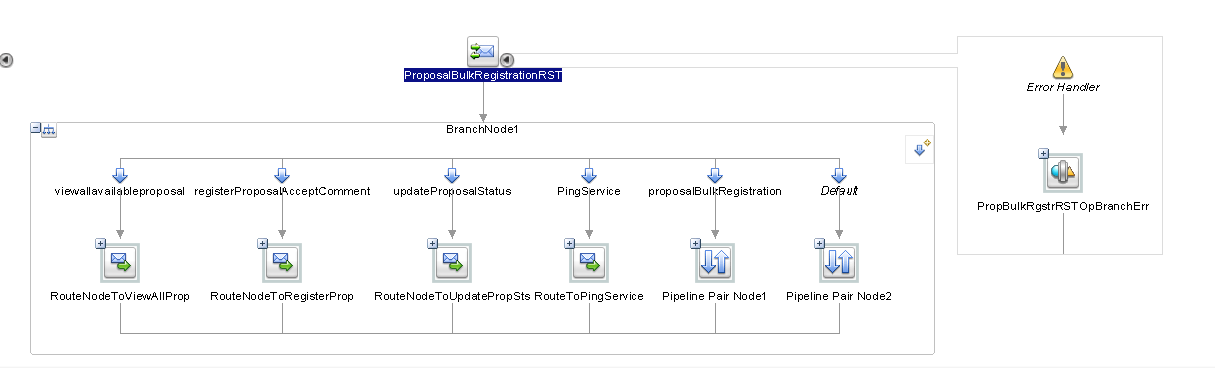
</resource>

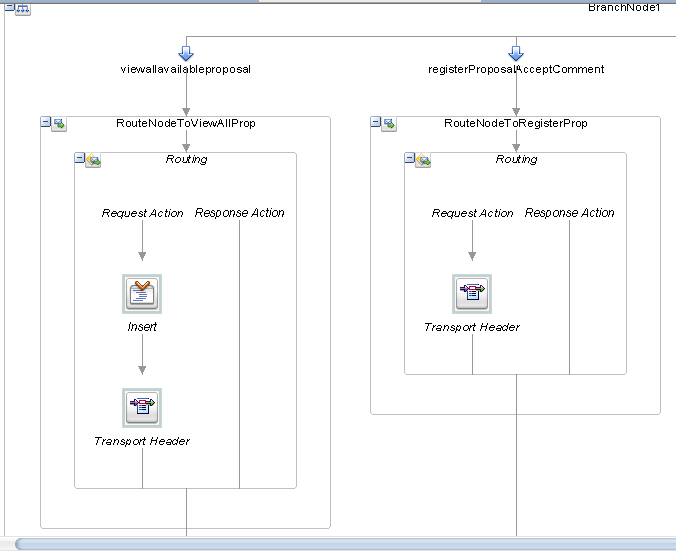
</resources>

</application>

### 1.1.3 Creating Pipeline and Flow configuration

Use operational branching and then we can use Routing/Pipeline as per need.



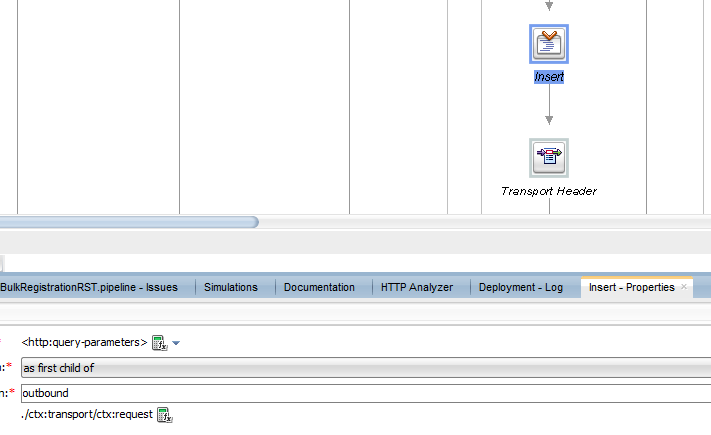


### 1.1.4 Setting Query Parameter

For Invoking GET operation we have to setup both “Query Parameter” and “Content-Type, Accept values” while for invoking POST,PUT operation we just need to setup “Content-Type, Accept values”. While invoking POST/PUT operations request data will be automatically passed into body-body, explicitly we need not to do any kind of transformation unlike SOAP call.

<http:query-parameters>

<http:parameter name="proposalId" value="{$inbound/ctx:transport/ctx:request/http:query-parameters/http:parameter[@name='proposalId']/@value}"/>

</http:query-parameters>

## Setting Header properties while invoking REST Svc

There are 2 ways to set header properties (**json or xml** value) while invoking REST Service:-

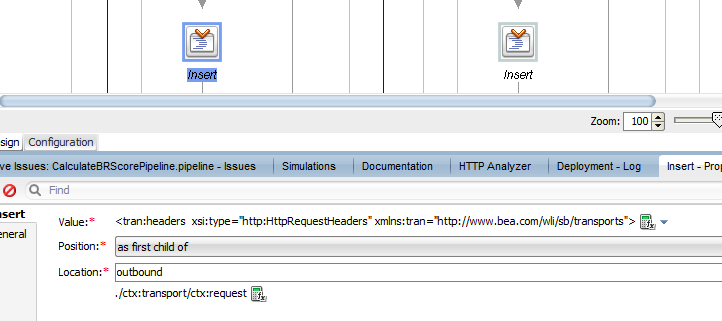
### 1.2.1 Using INSERT Activity

<tran:headers xsi:type="http:HttpRequestHeaders" xmlns:tran="http://www.bea.com/wli/sb/transports">

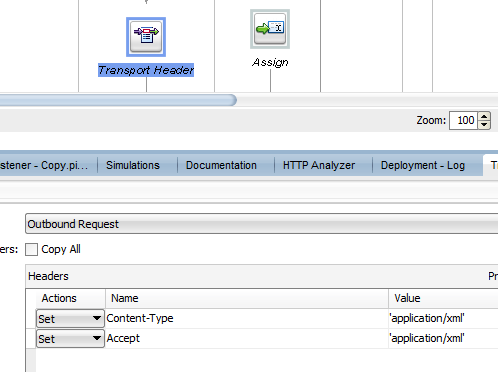
<http:Content-Type>application/xml;charset=UTF-8</http:Content-Type>

<http:Accept>application/xml;charset=UTF-8</http:Accept>

</tran:headers>



### Using TRANSPORT HEADER Activity

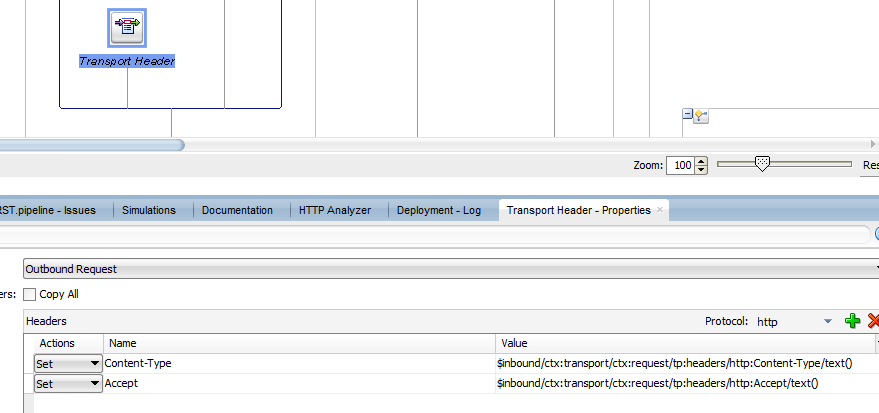


### 1.2.3 Using (Pass Through) TRANSPORT HEADER Activity

It will pass same value(**application/json** or **application/xml**) what will be requested from calling Svc-

Content-Type==$inbound/ctx:transport/ctx:request/tp:headers/http:Content-Type/text()

Accept==$inbound/ctx:transport/ctx:request/tp:headers/http:Accept/text()



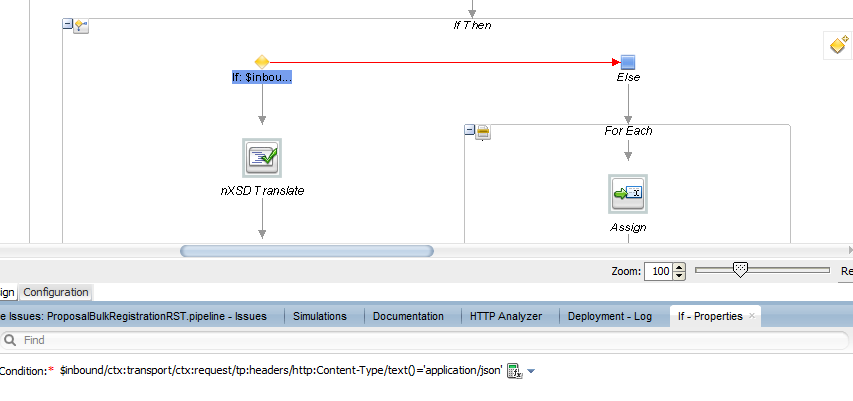
## Message Flow Orchestration

### 1.3.1 Assign incoming message body into **OriginalRequest** variable.

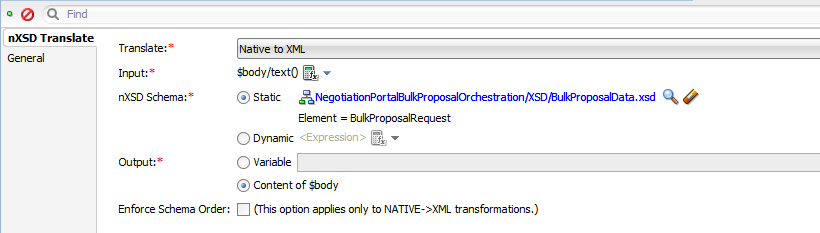
Assign Incoming message body into a variable.

If required to check what is incoming message format (JSON or XML)

$inbound/ctx:transport/ctx:request/tp:headers/http:Content-Type/text()='application/json'



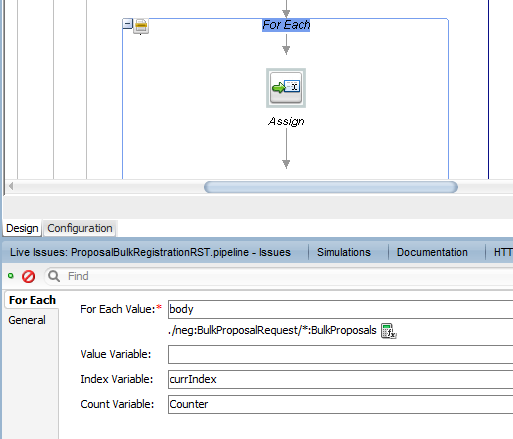
If JSON is the data then do transformation into XML using nXSD Translate.



After translation I assigned value of body into **OriginalRequest** variable.

### 1.3.2 FOR EACH Use

If you want to Iterate through each node in an array of nodes in xml then use FOR EACH activity-



### 1.3.3 Fetch value from Array of Element node

Use Assign Activity and do following:-

<neg:ProposalDataRequest xmlns:neg="http://www.multiplan.com/BulkProposalXSD">

<proposalId>{$OriginalRequest//\*:BulkProposals[xs:int($currIndex)]/\*:proposalId/text()}</proposalId> <claimNumber>{$OriginalRequest//\*:BulkProposals[xs:int($currIndex)]/\*:claimNumber/text()}</claimNumber>

<providerId>{$OriginalRequest//\*:providerId/text()}</providerId>

<negotiationProductId>{$OriginalRequest//\*:negotiationProductId/text()}</negotiationProductId>

<receivedFromLastName>{$OriginalRequest//\*:lastName/text()}</receivedFromLastName>

<receivedFromFirstName>{$OriginalRequest//\*:firstName/text()}</receivedFromFirstName>

<receivedFromEmailAddress>{$OriginalRequest//\*:email/text()}</receivedFromEmailAddress>

<receivedFromPhone>{$OriginalRequest//\*:phone/text()}</receivedFromPhone>

<receivedFromTitle>{$OriginalRequest//\*:title/text()}</receivedFromTitle>

<receivedFromComments>{$OriginalRequest//\*:comment/text()}</receivedFromComments>

<esignature>{$OriginalRequest//\*:digitalSignature/text()}</esignature>

<interestedInGlobal>{$OriginalRequest//\*:interestedInGlobal/text()}</interestedInGlobal>

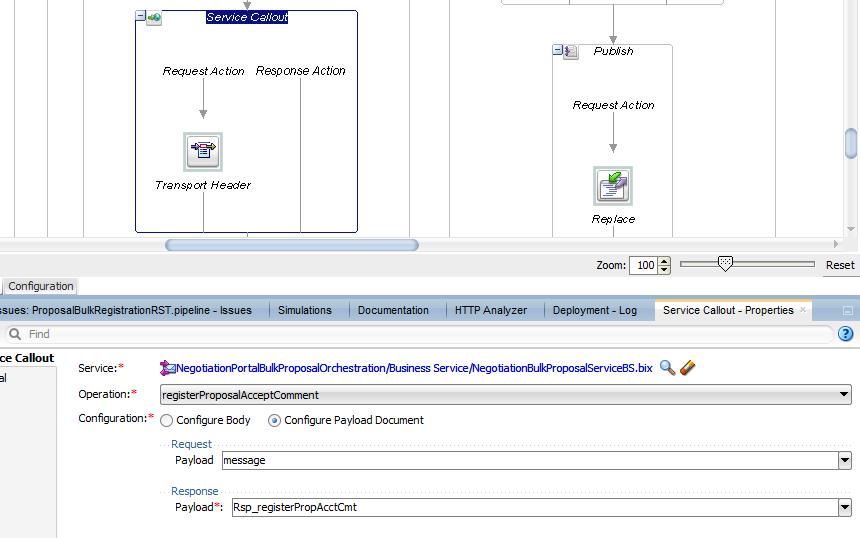
<documentMediaTypeId>2</documentMediaTypeId> <preferredComMethod>{$OriginalRequest//\*:preferredCommunicationMethod/text()}</preferredComMethod>

<proposalAction>{$OriginalRequest//\*:action/text()}</proposalAction>

</neg:ProposalDataRequest>

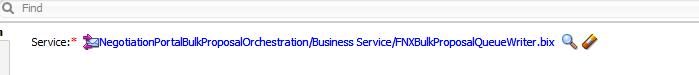
### 1.3.4 Service Call OUT

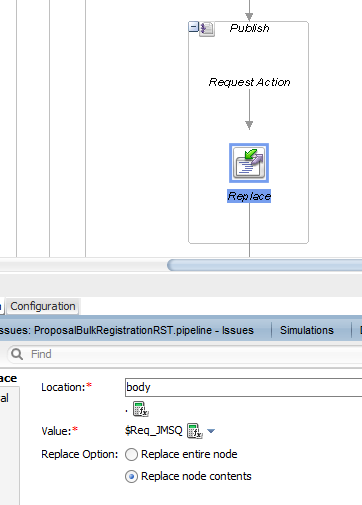
Select “Configure Payload Document” and pass created message for request and in Response use any variable other than context variable. Pass Accept value(Here I pass ‘application/xml’) using Transport Header Activity.



### 1.3.5 Publish Activity

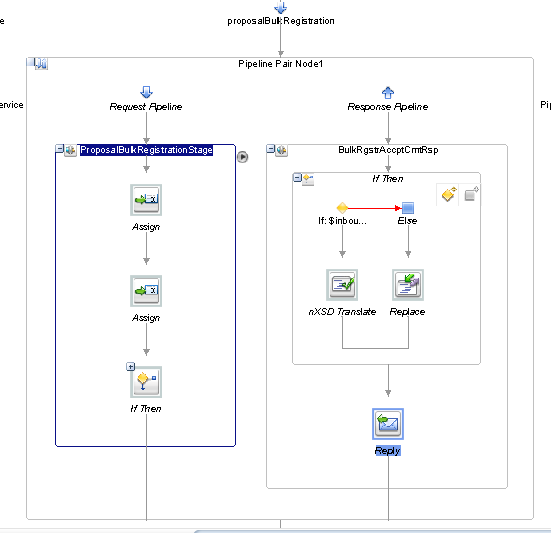
While invoking do a Replace activity and pass the request payload as following.





### 1.3.6 Response Pipeline(For REST Service)

When you have to respond back to invoking service then you need to configure some below properties



First check what is the Accept format(JSON or XML) for Invoking service.

$inbound/ctx:transport/ctx:request/tp:headers/http:Accept/text()='application/json'

If JSON then do nXSD Translation from XML to JSON and put it into body.

Input:-

<ns0:BulkProposalResponse xmlns:ns0="http://www.multiplan.com/BulkProposalXSD" xmlns:ns1="http://www.multiplan.com/xmlschema/status">

<ns1:RequestStatus>

<Status>

<StatusCD>SUCCESS</StatusCD>

</Status>

<Advice>

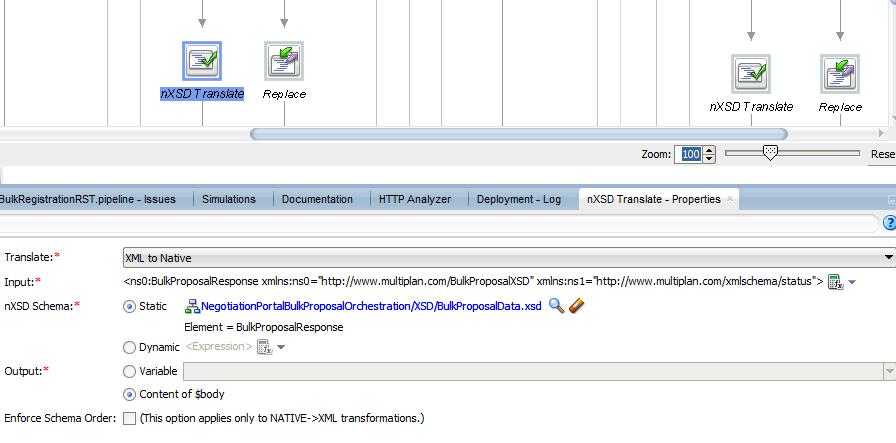
<AdviceCD>P</AdviceCD>

<AdviceComment/>

</Advice>

</ns1:RequestStatus>

</ns0:BulkProposalResponse>



If Accept was ‘application/xml’ then just do a Replace into body using xml payload message as below.

<ns0:BulkProposalResponse xmlns:ns0="http://www.multiplan.com/BulkProposalXSD" xmlns:ns1="http://www.multiplan.com/xmlschema/status">

<ns1:RequestStatus>

<Status>

<StatusCD>SUCCESS</StatusCD>

</Status>

<Advice>

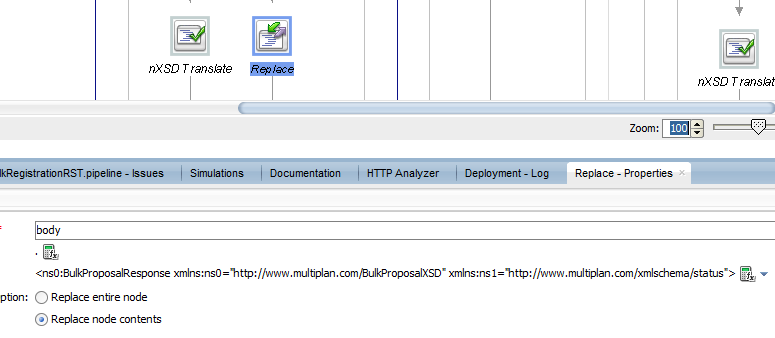
<AdviceCD>P</AdviceCD>

<AdviceComment/>

</Advice>

</ns1:RequestStatus>

</ns0:BulkProposalResponse>

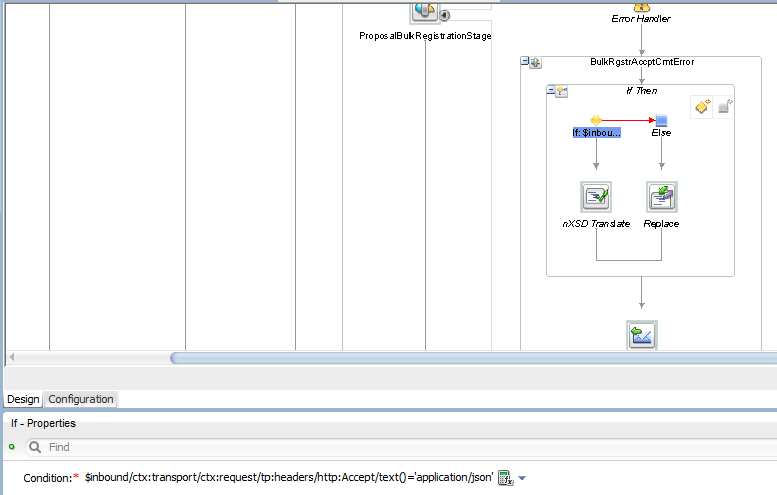


At the end use a Reply Activity with Success.

## Error Handling

### 1.4.1 Request Pipeline Level

Add Error Handler in request pipeline if any error happens then what we should respond back the client



Check via a condition if Accept is of JSON or XML?

If JSON then do nXSD Translate from xml to JSON and put value into body.

Input Content:-

<ns0:BulkProposalResponse xmlns:ns0="http://www.multiplan.com/BulkProposalXSD" xmlns:ns1="http://www.multiplan.com/xmlschema/status">

<ns1:RequestStatus>

<Status>

<StatusCD>FAILED</StatusCD>

</Status>

<Advice>

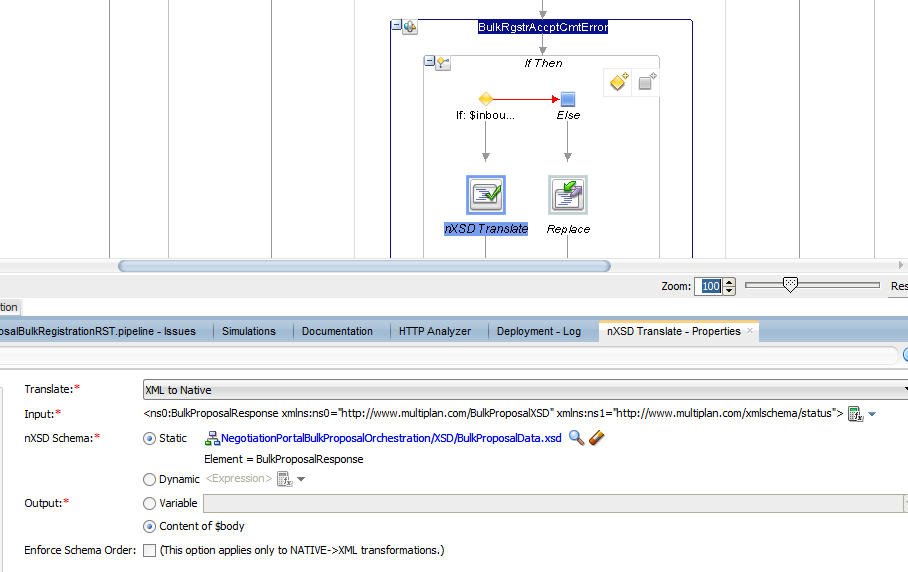
<AdviceCD>R</AdviceCD>

<AdviceComment>RESUBMIT</AdviceComment>

</Advice>

</ns1:RequestStatus>

</ns0:BulkProposalResponse>



If Accept is XML then just use a Replace activity and put xml into body as below

<ns0:BulkProposalResponse xmlns:ns0="http://www.multiplan.com/BulkProposalXSD" xmlns:ns1="http://www.multiplan.com/xmlschema/status">

<ns1:RequestStatus>

<Status>

<StatusCD>FAILED</StatusCD>

</Status>

<Advice>

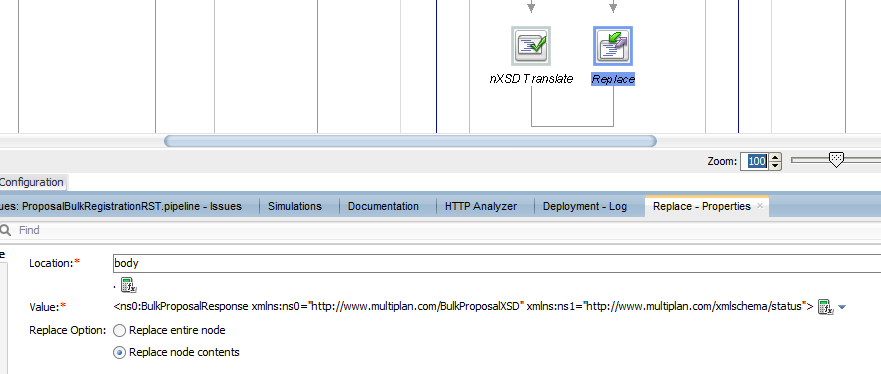
<AdviceCD>R</AdviceCD>

<AdviceComment>RESUBMIT</AdviceComment>

</Advice>

</ns1:RequestStatus>

</ns0:BulkProposalResponse>

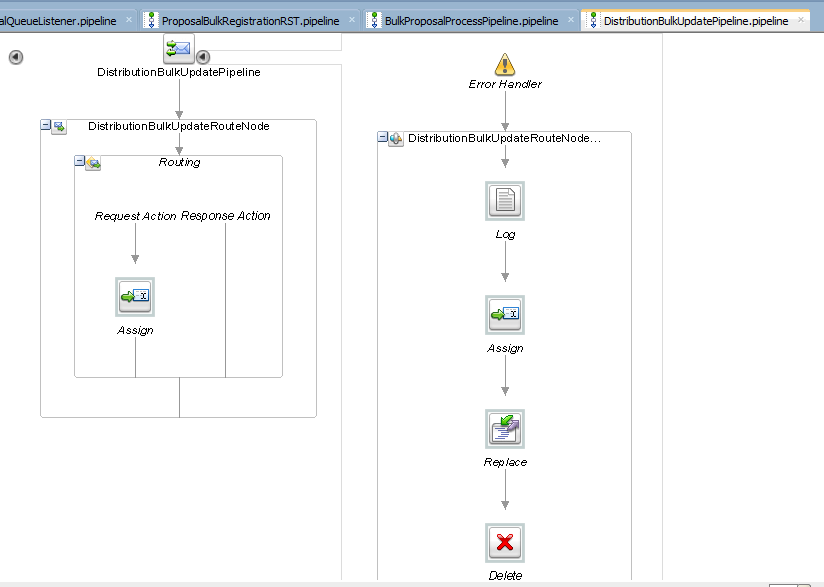
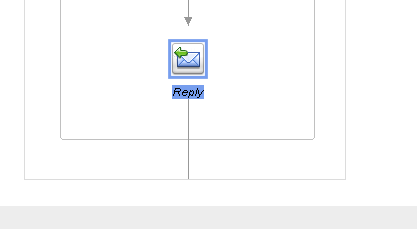


At the end Use a REPLY activity and reply with ‘Failure’.

### 1.4.2 Whole Message Flow Level

This is also called System level error handler.

#### 1.4.2.1 While invoking SOAP Services



Log the error:-

<failedMessage>

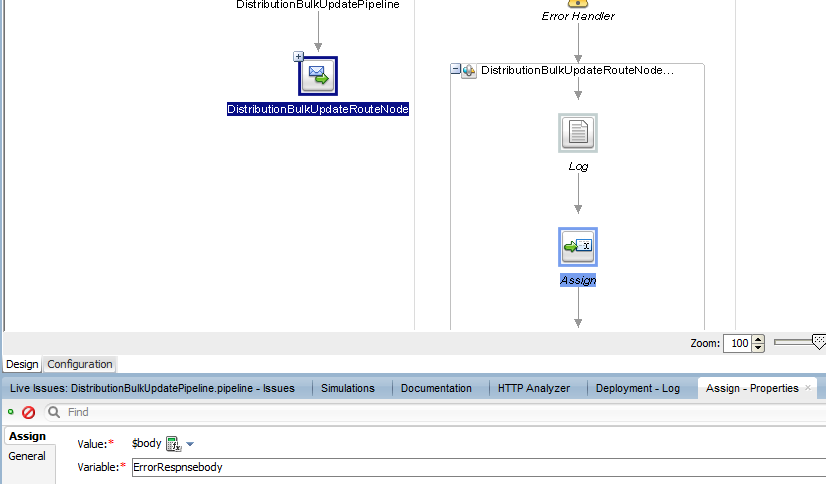
<failureReason>DistributionBulkUpdateRouteNode Failed</failureReason>

<IncomingRequest>{$OriginalRequest}</IncomingRequest>

<Message>{$body}</Message>

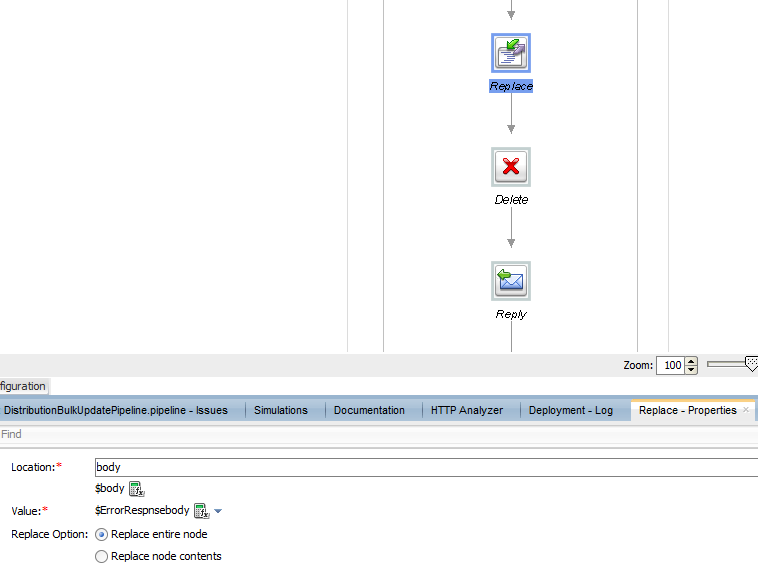
<ActualFault>{$fault}</ActualFault>

</failedMessage>

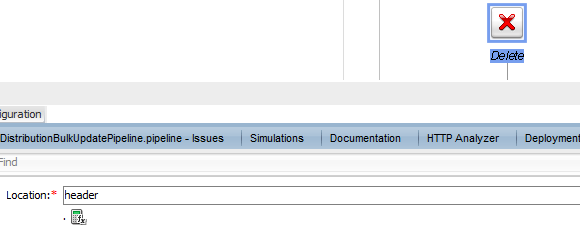


Assign body into a variable.

And do replace-

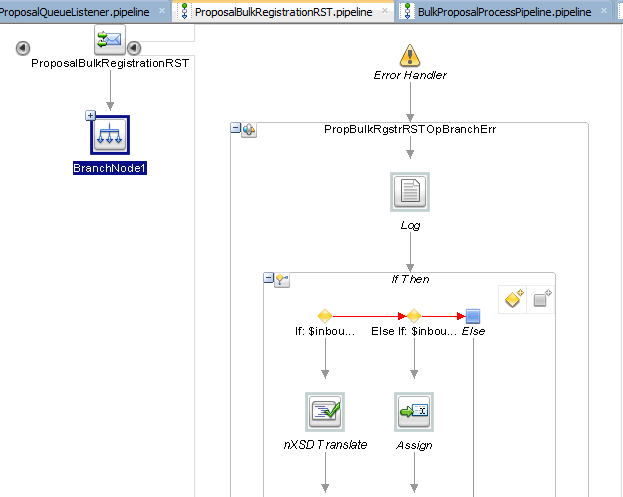
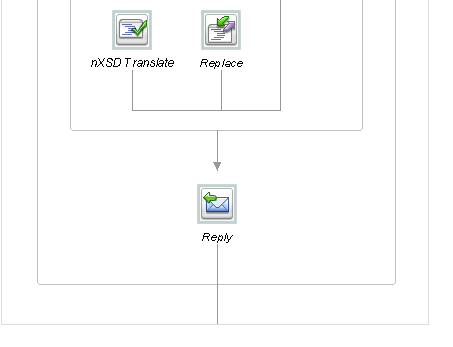


Delete the Header.



At the end reply with Failure.

#### 1.4.2.2 While invoking REST Services



ProposalBulkRegistration

Operation was not part of any

external service. This was created and

exposed from OSB itself as a REST

operation.That is in error handler

I checked if request format was JSON

Then do transformation otherwise just

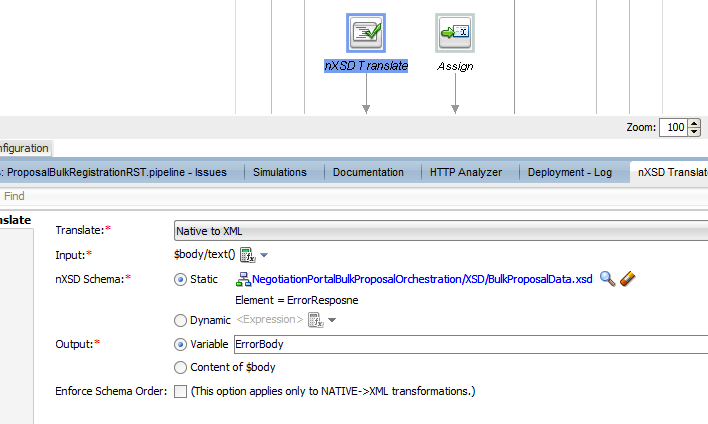
Assign and replace of error message

Was fine.

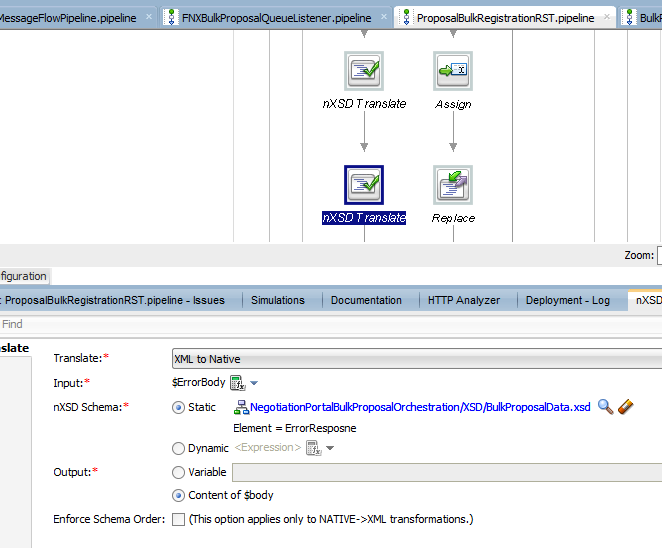
Why I did 2 times nXSD Translate, because at the time of error I was getting error message as JSON in CDATA that is why transformed it’s content into xml then again into JSON to get proper JSON format.

IF-> $inbound/ctx:transport/ctx:request/tp:headers/http:Accept/text()='application/json'

1st nXSD Translate-

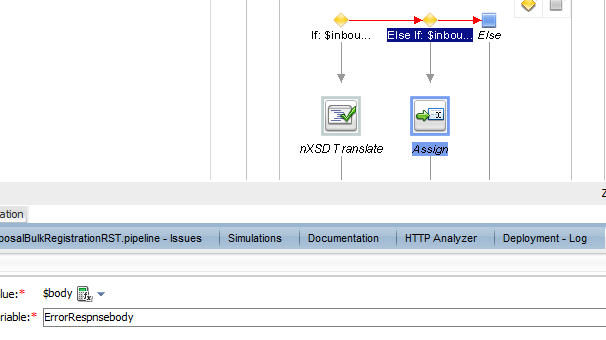


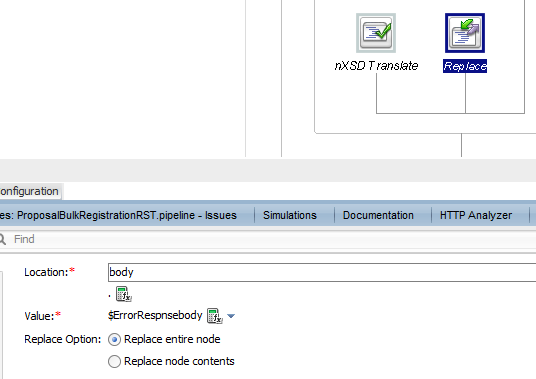
2nd nXSD Translate-



Else-If-> $inbound/ctx:transport/ctx:request/tp:headers/http:Accept/text()='application/xml'

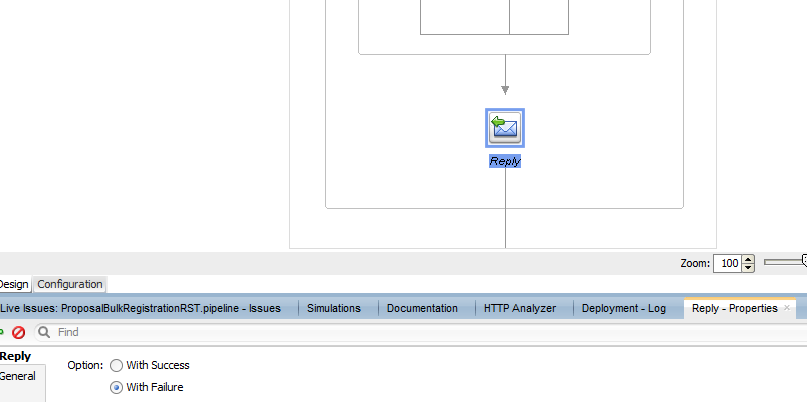
Assign body into a variable.





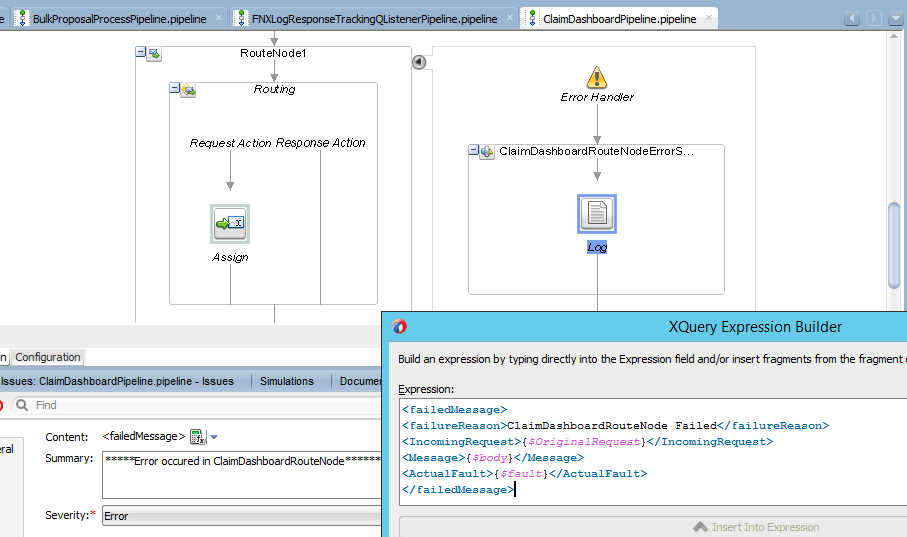
Use replace activity.

At the end use a Reply Activity with Failure.

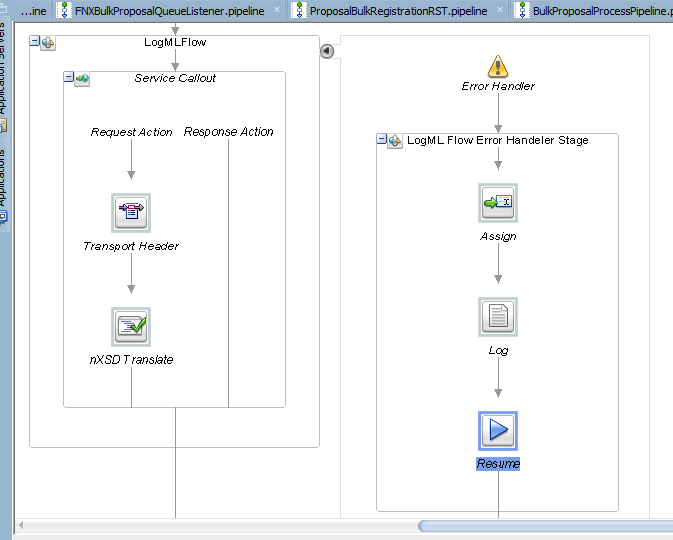


### 1.4.3 Routing Level

Use Error handler and use Log activity to log the details as below.



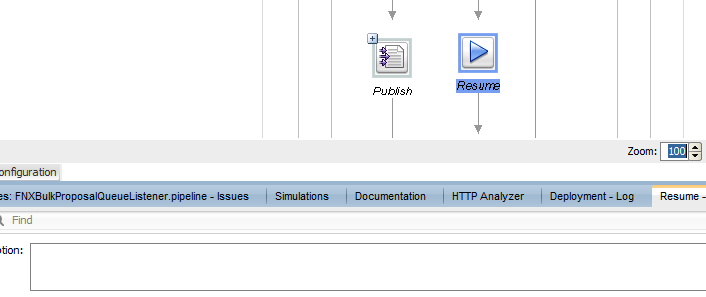
### 1.4.4 Stage Level



## Important Activities

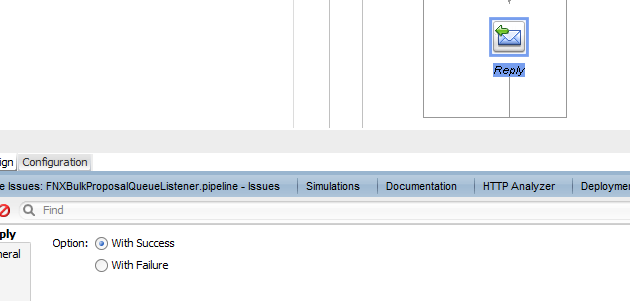
### 1.5.1 **RESUME**

If you want to keep continuing the flow even if it fails then we need to use Resume Activity inside Error Handler.



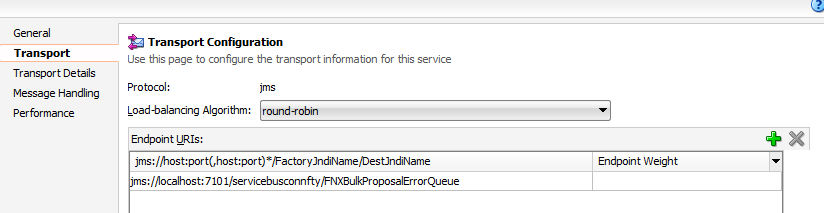
### 1.5.2 **REPLY**

When you want to reply to client either “Success” or “Failure” then we can use Reply activity in Response Pipeline or inside Error Handler.



### 1.5.3 JMS Transport component

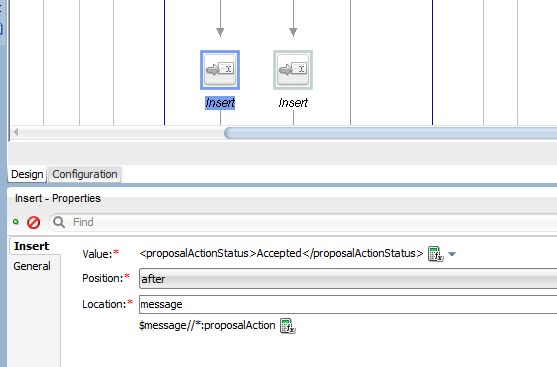
Read or write data into JMS Q we can use JMS Transport too.



### 1.5.4 INSERT

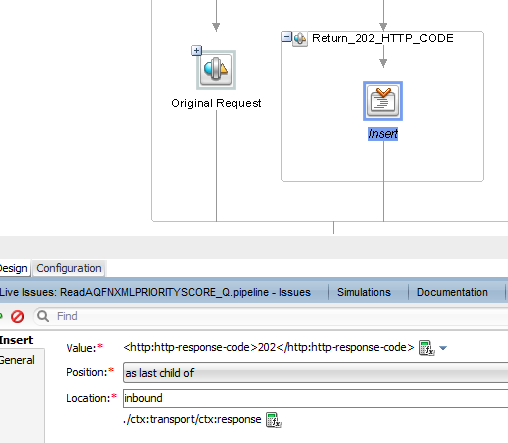
Is used to insert xml element in an xml payload.

#### 1.5.4.1 Inserting into Variable XML Payload



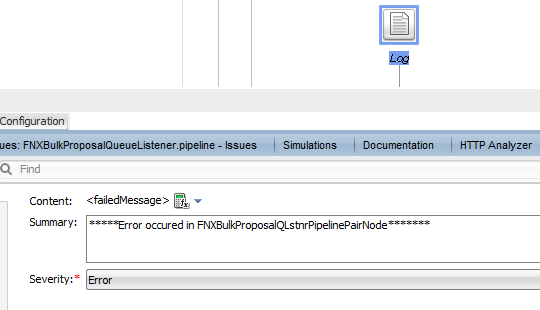
#### 1.5.4.2 Inserting into Context Variable XML Payload

Below I added Http-response-code inside “Inbound” context variable.



### 1.5.5 LOG Activity

#### 1.5.5.1 Log SOAP Type error details in LOG Activity



Content:-

<failedMessage>

<failureReason>FNXBulkProposalQLstnrPipelinePairNode Failed</failureReason>

<IncomingRequest>{$OriginalRequest}</IncomingRequest>

<Message>{$body}</Message>

<ActualFault>{$fault}</ActualFault>

</failedMessage>

#### 1.5.5.2 Log REST Type error details in LOG Activity

<failedMessage>

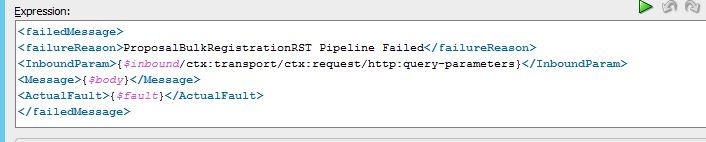
<failureReason>ProposalBulkRegistrationRST Pipeline Failed</failureReason>

<InboundParam>{$inbound/ctx:transport/ctx:request/http:query-parameters}</InboundParam>

<Message>{$body}</Message>

<ActualFault>{$fault}</ActualFault>

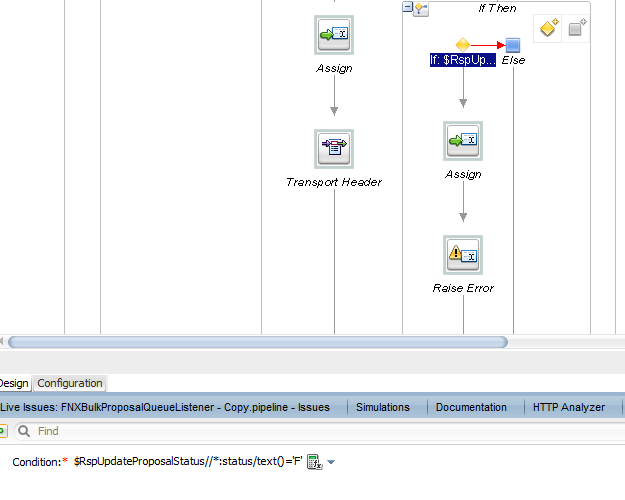
</failedMessage>

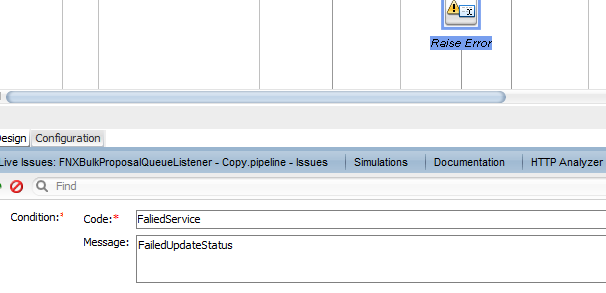


### 1.5.6 RAISE ERROR

If you want to raise business errors inside the flow then we can use Raise-error activity.

Use any condition then raise the error

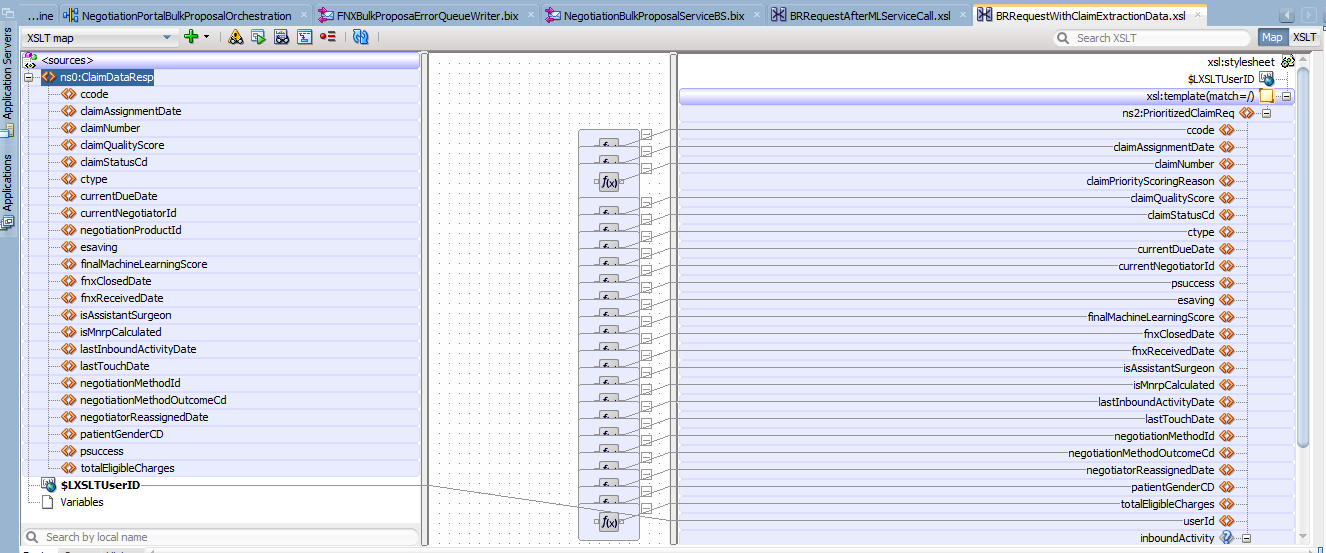


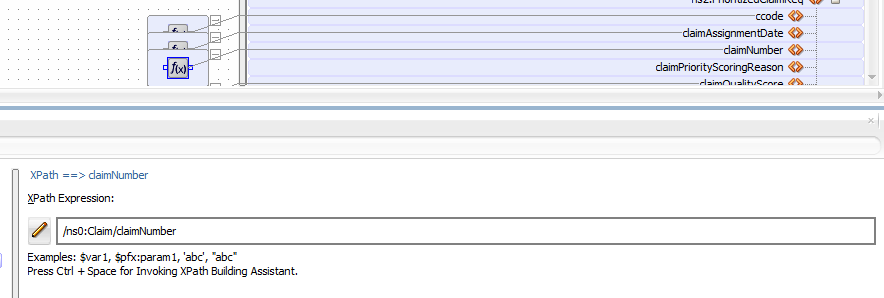


### 1.5.7 XSLT Transformation

Keep in mind Source xml Payload (Element names) doesn’t matter at run time for xslt transformation.

Here below I have defined source payload will have Root element as “**ClaimDataResp**” but I am passing “Claim” as root element.So during Transformation in XSLT source code use the element x-path same what you will be passing at run time during xslt invocation.





And if you have declared any variable during XSLT configuration then you will have to pass its value during invocation.

BRRequestAfterMLServiceCall.xsl—

<xsl:stylesheet version="1.0" exclude-result-prefixes=" xsd oracle-xsl-mapper xsi xsl ns0 ns1 ns2 UUIDUserFunction IsUserInGroupFunction oraext IsUserInRoleFunction xp20 DVMFunctions oraxsl RuntimeTypeConversionFunctions XrefFunctions BasicCredentialsUserFunction" xmlns:UUIDUserFunction="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.stages.functions.UUIDUserFunction" xmlns:IsUserInGroupFunction="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.stages.functions.IsUserInGroupFunction" xmlns:ns1="http://Multiplan.com/NegotiationPrioritization" xmlns:oraext="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.ExtFunc" xmlns:IsUserInRoleFunction="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.stages.functions.IsUserInRoleFunction" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functions.Xpath20" xmlns:DVMFunctions="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.functions.dvm.DVMFunctions" xmlns:oracle-xsl-mapper="http://www.oracle.com/xsl/mapper/schemas" xmlns:oraxsl="http://www.oracle.com/XSL/Transform/java" xmlns:RuntimeTypeConversionFunctions="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.stages.functions.RuntimeTypeConversionFunctions" xmlns:XrefFunctions="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.functions.xref.XrefFunctions" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns0="http://Multiplan.com/NegotiationClaimData" xmlns:ns2="http://Multiplan.com/MachineLearningScoreData" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:BasicCredentialsUserFunction="http://www.oracle.com/XSL/Transform/java/com.bea.wli.sb.stages.functions.BasicCredentialsUserFunction" xmlns:nxsd="http://xmlns.oracle.com/pcbpel/nxsd">

<oracle-xsl-mapper:schema>

<!--SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY.-->

<oracle-xsl-mapper:mapSources>

<oracle-xsl-mapper:source type="XSD">

<oracle-xsl-mapper:schema location="../../ResourcesNegSvc/xsd/ClaimPrioritization/NegotiationClaimData.xsd"/>

<oracle-xsl-mapper:rootElement name="ClaimDataResp" namespace="http://Multiplan.com/NegotiationClaimData"/>

</oracle-xsl-mapper:source>

<oracle-xsl-mapper:source type="XSD">

<oracle-xsl-mapper:schema location="../../ResourcesNegSvc/xsd/ClaimPrioritization/NegotiationML.xsd"/>

<oracle-xsl-mapper:rootElement name="MLScoreDataResp" namespace="http://Multiplan.com/MachineLearningScoreData"/>

<oracle-xsl-mapper:param name="LMLResp"/>

</oracle-xsl-mapper:source>

</oracle-xsl-mapper:mapSources>

<oracle-xsl-mapper:mapTargets>

<oracle-xsl-mapper:target type="XSD">

<oracle-xsl-mapper:schema location="../../ResourcesNegSvc/xsd/ClaimPrioritization/NegotiationPrioritization.xsd"/>

<oracle-xsl-mapper:rootElement name="PrioritizedClaimReq" namespace="http://Multiplan.com/NegotiationPrioritization"/>

</oracle-xsl-mapper:target>

</oracle-xsl-mapper:mapTargets>

<!--GENERATED BY ORACLE XSL MAPPER 12.2.1.2.0(XSLT Build 161003.0739.0018) AT [MON FEB 10 07:35:17 EST 2020].-->

</oracle-xsl-mapper:schema>

<!--User Editing allowed BELOW this line - DO NOT DELETE THIS LINE-->

<xsl:param name="LXSLTUserID"/>

<xsl:param name="LMLResp"/>

<xsl:template match="/">

<ns1:PrioritizedClaimReq>

<ccode>

<xsl:value-of select="//ns0:Claim/ccode"/>

</ccode>

<claimAssignmentDate>

<xsl:value-of select="/ns0:Claim/claimAssignmentDate"/>

</claimAssignmentDate>

<claimNumber>

<xsl:value-of select="/ns0:Claim/claimNumber"/>

</claimNumber>

<claimPriorityScoringReason/>

<claimQualityScore>

<xsl:value-of select="/ns0:Claim/claimQualityScore"/>

</claimQualityScore>

<claimStatusCd>

<xsl:value-of select="/ns0:Claim/claimStatusCd"/>

</claimStatusCd>

<ctype>

<xsl:value-of select="/ns0:Claim/ctype"/>

</ctype>

<currentDueDate>

<xsl:value-of select="/ns0:Claim/currentDueDate"/>

</currentDueDate>

<currentNegotiatorId>

<xsl:value-of select="/ns0:Claim/currentNegotiatorId"/>

</currentNegotiatorId>

<psuccess>

<xsl:value-of select="$LMLResp//\*:machineLearningModelValues/psuccess"/>

</psuccess>

<esaving>

<xsl:value-of select="$LMLResp//\*:machineLearningModelValues/esavings"/>

</esaving>

<finalMachineLearningScore>

<xsl:value-of select="$LMLResp//\*:machineLearningModelValues/score"/>

</finalMachineLearningScore>

<fnxClosedDate>

<xsl:value-of select="/ns0:Claim/fnxClosedDate"/>

</fnxClosedDate>

<fnxReceivedDate>

<xsl:value-of select="/ns0:Claim/fnxReceivedDate"/>

</fnxReceivedDate>

<isAssistantSurgeon>

<xsl:value-of select="/ns0:Claim/isAssistantSurgeon"/>

</isAssistantSurgeon>

<isMnrpCalculated>

<xsl:value-of select="/ns0:Claim/isMnrpCalculated"/>

</isMnrpCalculated>

<lastInboundActivityDate>

<xsl:value-of select="/ns0:Claim/lastInboundActivityDate"/>

</lastInboundActivityDate>

<lastTouchDate>

<xsl:value-of select="/ns0:Claim/lastTouchDate"/>

</lastTouchDate>

<negotiationMethodId>

<xsl:value-of select="/ns0:Claim/negotiationMethodId"/>

</negotiationMethodId>

<negotiationMethodOutcomeCd>

<xsl:value-of select="/ns0:Claim/negotiationMethodOutcomeCd"/>

</negotiationMethodOutcomeCd>

<negotiatorReassignedDate>

<xsl:value-of select="/ns0:Claim/negotiatorReassignedDate"/>

</negotiatorReassignedDate>

<patientGenderCD>

<xsl:value-of select="/ns0:Claim/patientGenderCD"/>

</patientGenderCD>

<totalEligibleCharges>

<xsl:value-of select="/ns0:Claim/totalEligibleCharges"/>

</totalEligibleCharges>

<userId>

<xsl:value-of select="$LXSLTUserID"/>

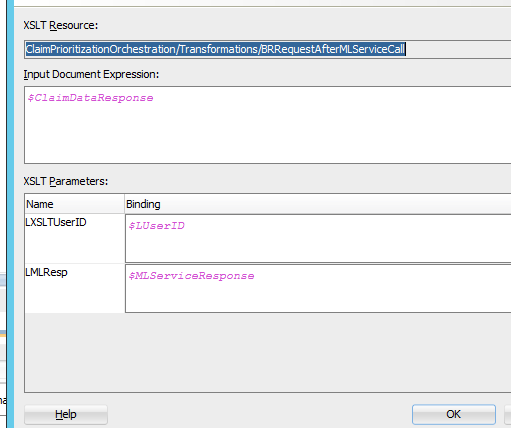
</userId>

</ns1:PrioritizedClaimReq>

</xsl:template>

</xsl:stylesheet>

Passing values during invocation:-



### 1.5.8 RemoveNamespace.xsl

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="xml" indent="no"/>

<xsl:template match="/|comment()|processing-instruction()">

<xsl:copy>

<xsl:apply-templates/>

</xsl:copy>

</xsl:template>

<xsl:template match="\*">

<xsl:element name="{local-name()}">

<xsl:apply-templates select="@\*|node()"/>

</xsl:element>

</xsl:template>

<xsl:template match="@\*">

<xsl:attribute name="{local-name()}">

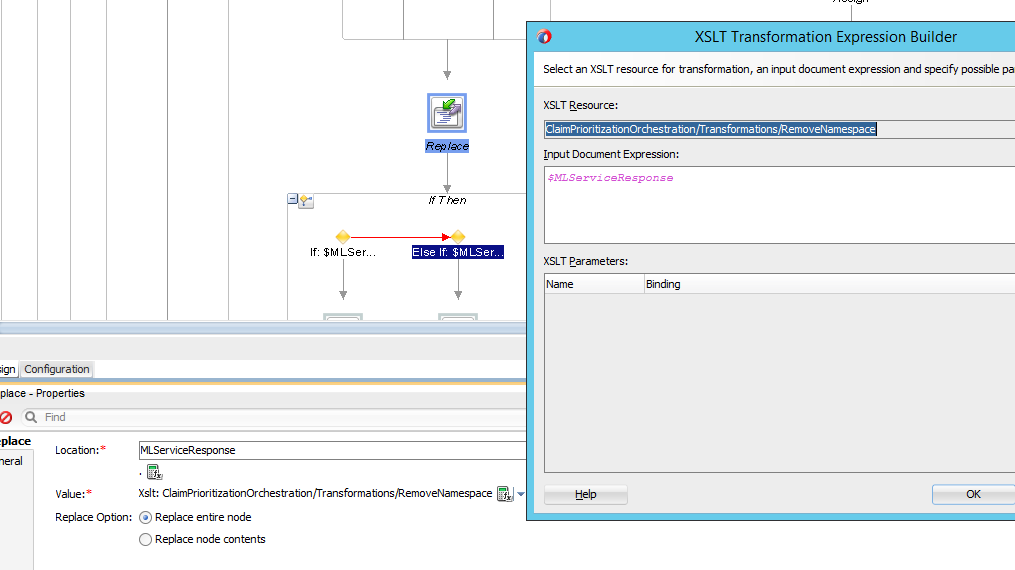
<xsl:value-of select="."/>

</xsl:attribute>

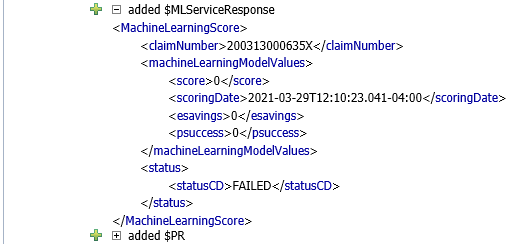
</xsl:template>

</xsl:stylesheet>

Invocation:-



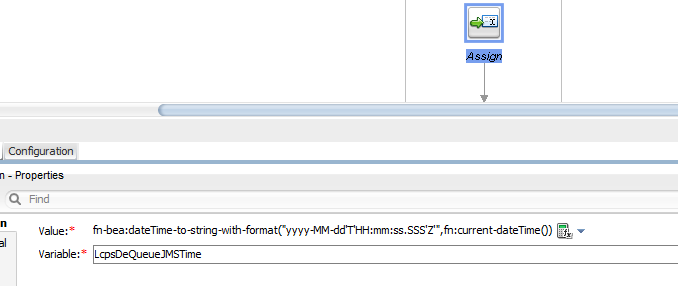
Before Removing Namespace After Removing Namespace



### 1.5.9 DateTime

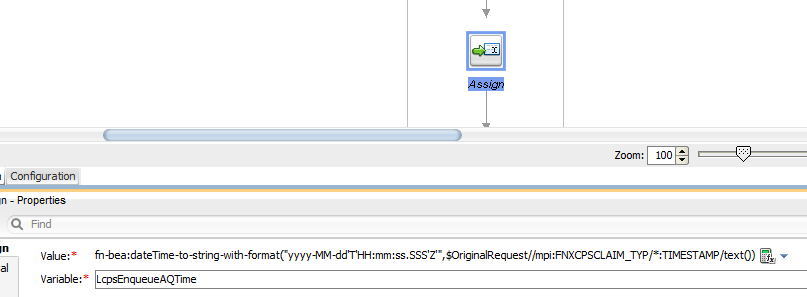
#### 1.5.9.1 Zulu Time Zone Date Time

fn-bea:dateTime-to-string-with-format("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",fn:current-dateTime())

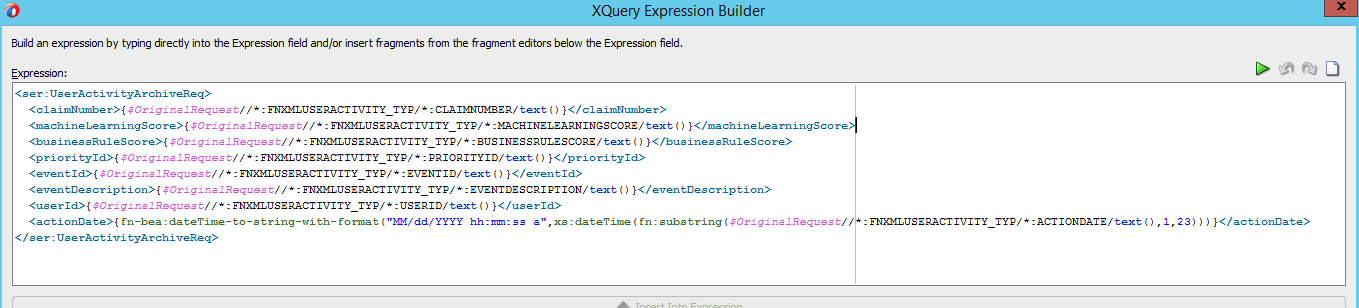


#### 1.5.9.2 Date Time to String

fn-bea:dateTime-to-string-with-format("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'",$OriginalRequest//mpi:FNXCPSCLAIM\_TYP/\*:TIMESTAMP/text())

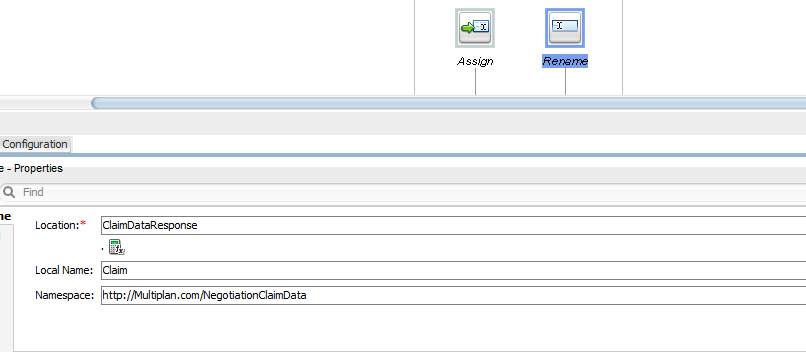


Another way to convert Oracle DB DateTime to XML-



### 1.5.10 RENAME Activity

There was no namespace attached to a response coming from service so to add a namespace I used Rename Activity. Because namespace was required for transformation.

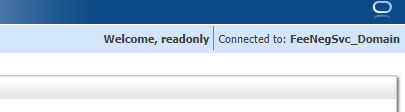


Before Transformation After Transformation

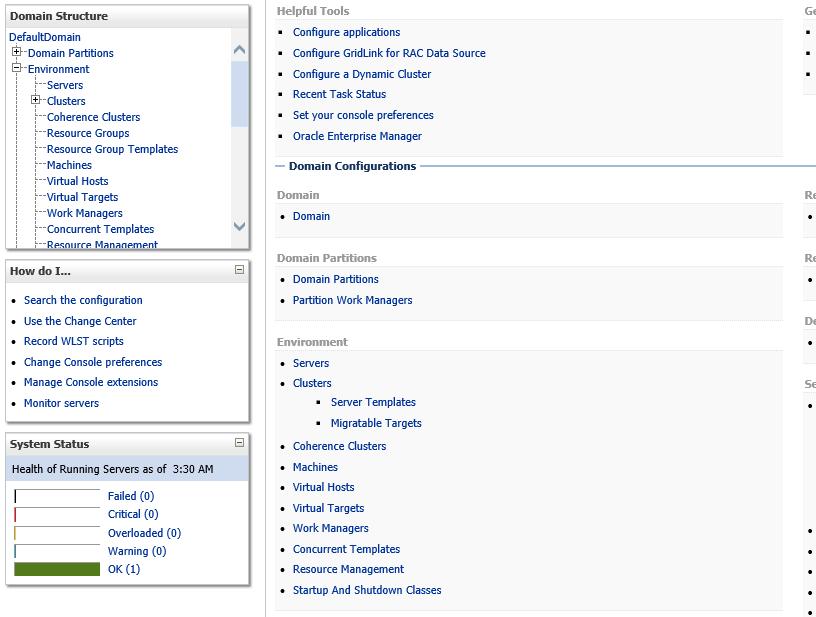


# 2. Managed Server Creation

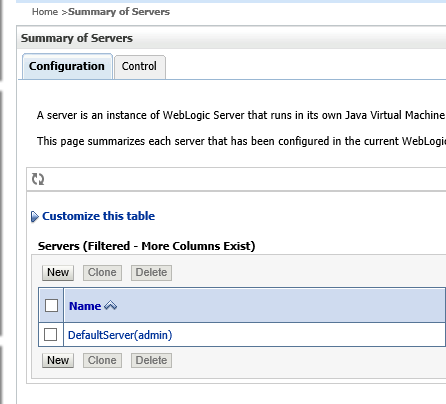
1. Login to Admin Console and connect to FeeNegSvc\_Domain.



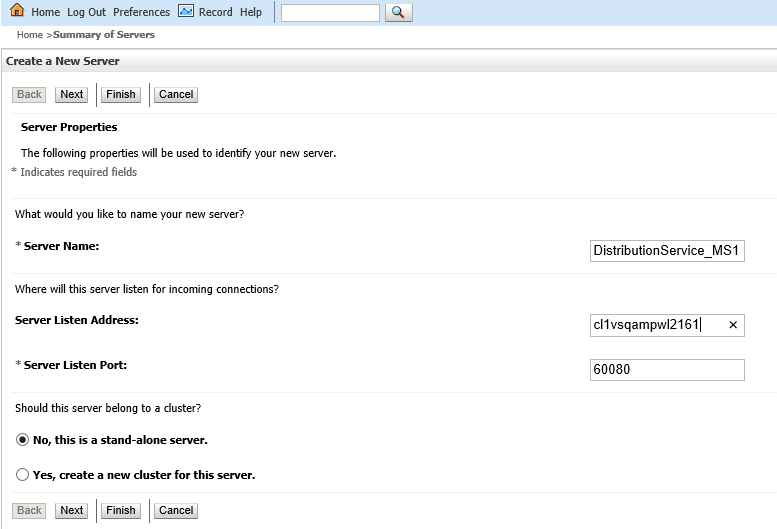
2. Go to Environment section and click on "Servers" option.



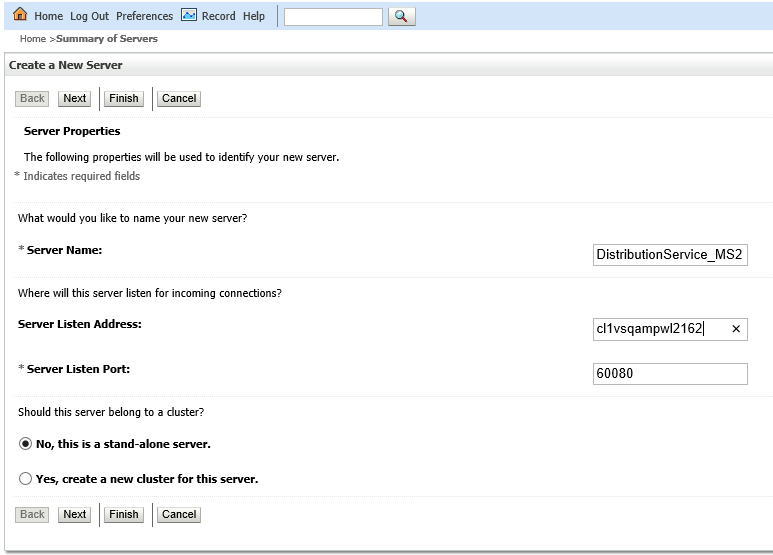
3. Click on new button to create new Managed Server.



4. Put name as "DistributionService\_MS1" and Server Listen Port as 60080 and Server Listen Address as "cl1vsqampwl2161"(for different environments it can be different.) Click Next and Finish.



5. Repeat step 3 and 4 for creating another managed server having name as "DistributionService\_MS2"



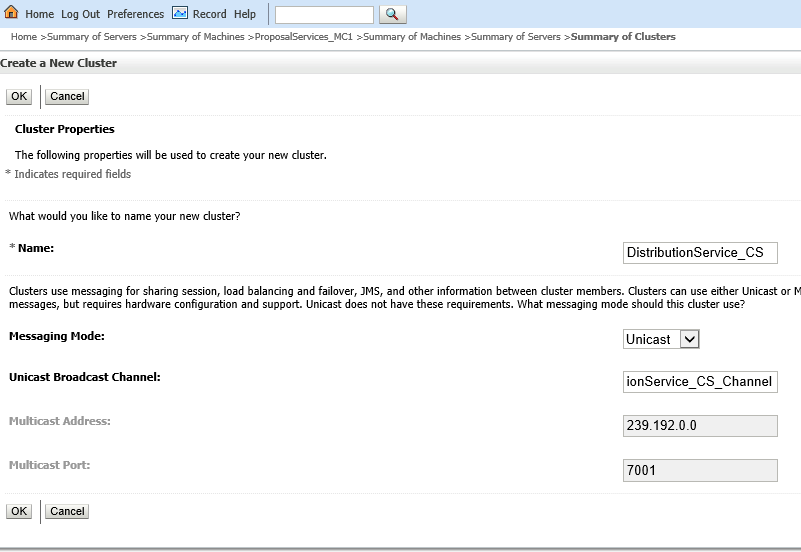
6.

* Managed Server Creation

Now we will create cluster. Go to Cluster option and click new and put the details as below.

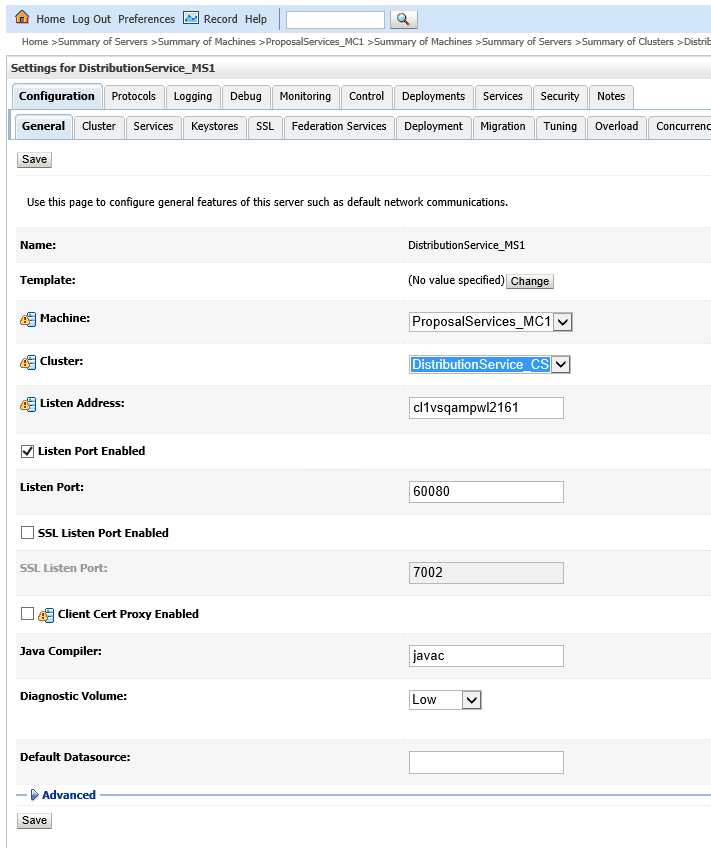
"DistributionService\_CS"

and Unicast Broadcast Channel Name as "DistributionService\_CS\_Channel". Click OK

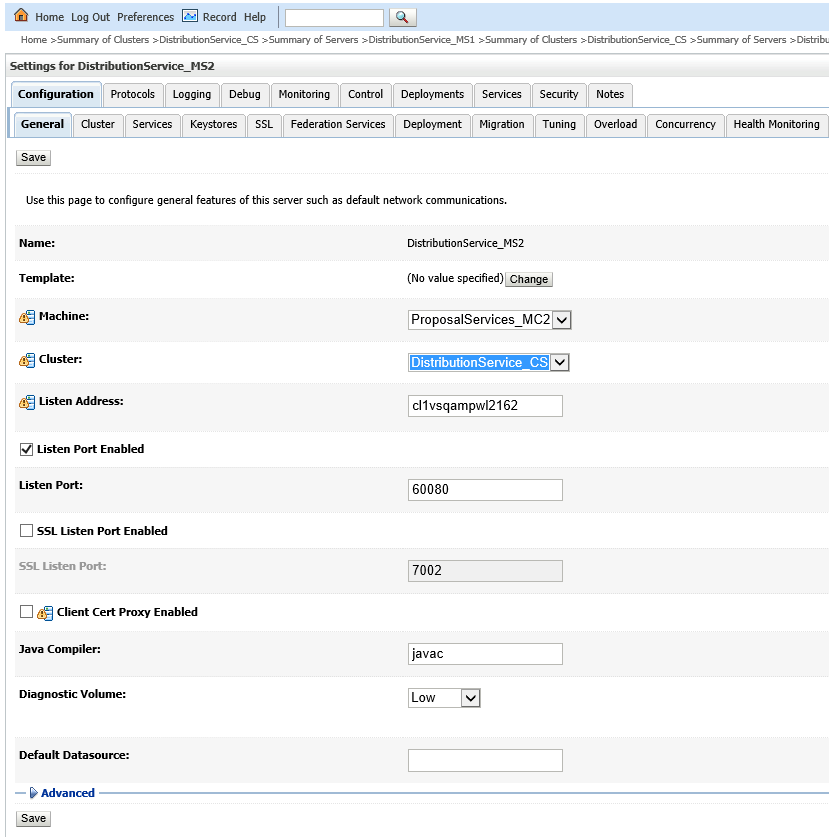


7. Now add Server instances into the cluster

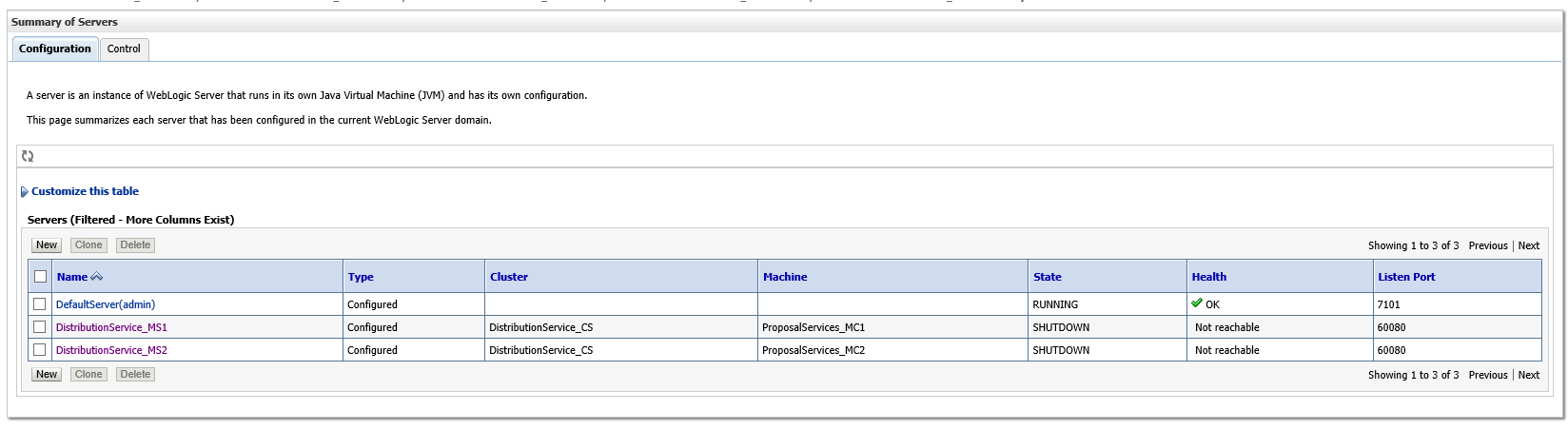
go to the "DistributionService\_MS1" Server and go to General and select machine as "ProposalServices\_MC1" and cluster as "DistributionService\_CS".



8. Do the same thing for the "DistributionService\_MS1" Server and go to General and select machine as "ProposalServices\_MC2" and cluster as "DistributionService\_CS".



9. This is the end of the definition of Cluster and Managed Server.



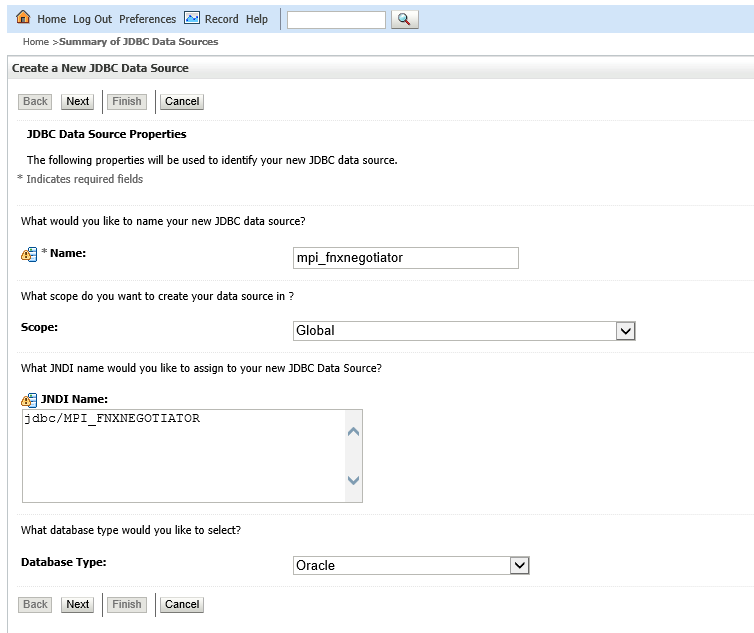
# 3.DataSource Creation

In this section we will create DataSource having name as "mpi\_ fnxnegotiator" and JNDI name as "jdbc/MPI\_FNXNEGOTIATOR".

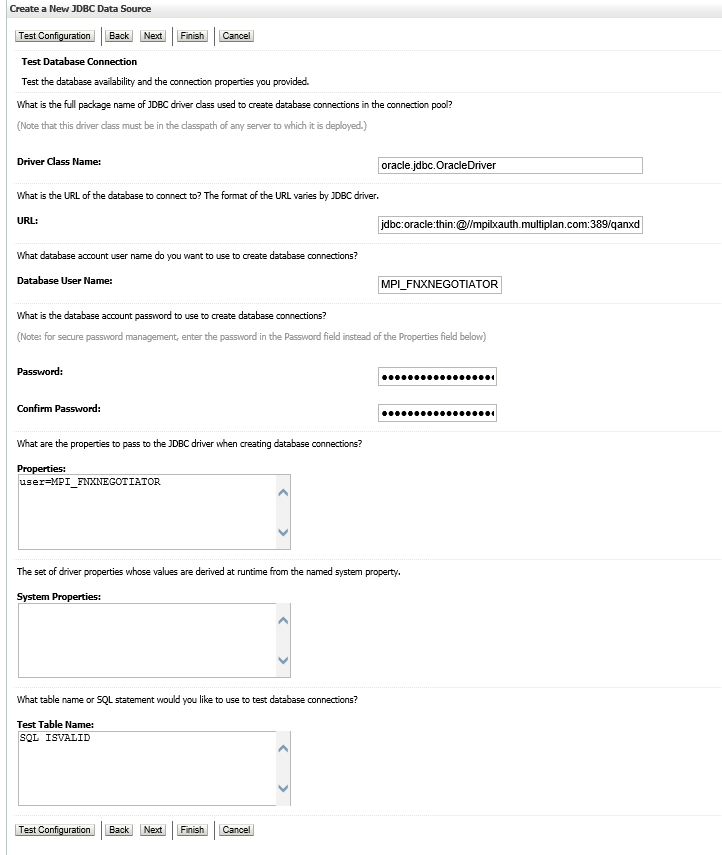
Steps:-

1. Go to DataSource Option and click on new and select Generic Datasource.

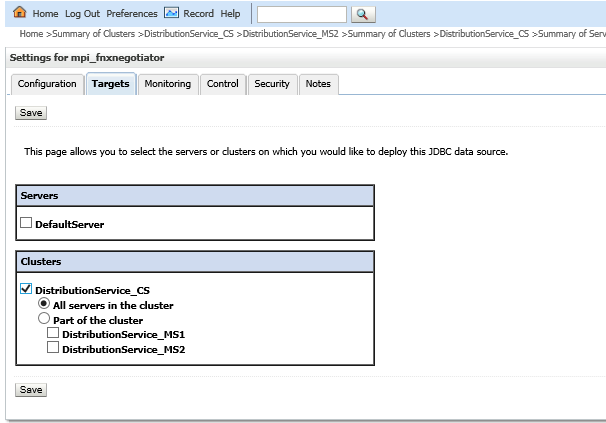
Fill the details as below.



2. Go to Next,Next and now put details for Database and Schema Password.



3. Go to Target tab and select the cluster.



4. Click save and you are done with the DataSource definitionDataSource Creation