

# CO-OP Bank

## ESB

# How To: Create BW Services from Templates

Document Version: v1.2.0

Document Date: April 2018



The Power of Now®

<http://www.TIBCO.com>

### Global Headquarters

3303 Hillview Avenue  
Palo Alto, CA 94304  
Tel: +1 650-846-1000  
Toll Free: 1 800-420-8450  
Fax: +1 650-846-1005

©Copyright 2010, TIBCO Software Inc. All rights reserved. TIBCO, the TIBCO logo, The Power of Now, and TIBCO Software are trademarks or registered trademarks of TIBCO Software Inc. in the United States and/or other countries. All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

## Copyright Notice

COPYRIGHT© 2010 TIBCO Software Inc. This document is unpublished and the foregoing notice is affixed to protect TIBCO Software Inc. in the event of inadvertent publication. All rights reserved. No part of this document may be reproduced in any form, including photocopying or transmission electronically to any computer, without prior written consent of TIBCO Software Inc. The information contained in this document is confidential and proprietary to TIBCO Software Inc. and may not be used or disclosed except as expressly authorized in writing by TIBCO Software Inc. Copyright protection includes material generated from our software programs displayed on the screen, such as icons, screen displays, and the like.

## Trademarks

Technologies described herein are either covered by existing patents or patent applications are in progress. All brand and product names are trademarks or registered trademarks of their respective holders and are hereby acknowledged.

## Confidentiality

The information in this document is subject to change without notice. This document contains information that is confidential and proprietary to TIBCO Software Inc. and may not be copied, published, or disclosed to others, or used for any purposes other than review, without written authorization of an officer of TIBCO Software Inc. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

## Content Warranty

The information in this document is subject to change without notice. THIS DOCUMENT IS PROVIDED "AS IS" AND TIBCO MAKES NO WARRANTY, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO ALL WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TIBCO Software Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

For more information, please contact:

TIBCO Software Inc.  
3303 Hillview Avenue  
Palo Alto, CA 94304  
USA

# Table of Contents

---

<b>1 Document Control .....</b>	<b>7</b>
1.1 Versioning Information .....	7
1.2 Distribution Information.....	7
1.2.1 CO-OP Bank .....	7
1.2.2 TIBCO .....	7
1.3 Reference Documents .....	7
1.4 Document Acceptance.....	8
<b>2 Introduction .....</b>	<b>9</b>
2.1 Scope.....	9
2.2 Purpose .....	9
2.3 Audience.....	9
2.4 Acronyms.....	9
2.5 Issues .....	10
2.6 Risks .....	10
<b>3 Introduction to the Templates.....</b>	<b>11</b>
<b>4 Create a Business Service Provider as an Application .....</b>	<b>12</b>
4.1 Template Overview.....	12
4.2 Pre-Requisites .....	12
4.2.1 Create the Application Module.....	12
4.2.2 Change the Module Properties .....	15
4.2.3 Rename the Resources .....	16
4.2.4 Create the Process Packages .....	16
4.2.5 Change the Service Process .....	17
4.2.6 Fix the Process Mappings.....	24
4.2.7 Create the SOAP Binding .....	32
4.3 Implement the Service .....	32
4.3.1 Business Logic.....	32
4.3.2 AssignResponse .....	32
4.3.3 Error Handling .....	32
4.3.4 Invoke another ESB service.....	34
4.3.5 Validate the Request.....	35
4.4 Finalization.....	37
<b>5 Create a Business Service Provider as a Shared Module .....</b>	<b>38</b>
5.1 Template Overview.....	38
5.2 Pre-Requisites .....	38
5.3 Procedure .....	38
5.3.1 Create the Application Module.....	38
5.3.2 Module Properties, Processes, Service Invocation.....	40
5.4 Finalization.....	40
<b>6 Create a Technical Service Provider as a Shared Module .....</b>	<b>41</b>
6.1 Template Overview .....	41
6.2 Pre-Requisites .....	41
6.3 Procedure .....	41

6.3.1	Create the Application Module .....	41
6.3.2	Change the Module Properties .....	44
6.3.3	Create the Process Packages .....	46
6.3.4	Change the Service Process .....	47
6.3.5	Fix the Process Mappings.....	51
6.3.6	Back End on HTTP/HTTPS .....	58
6.3.7	Back End on JDBC .....	66
6.4	Implement the Service .....	68
6.4.1	Service Response.....	68
6.4.2	Invoking another ESB service.....	68
6.4.3	Validate the Request.....	68
6.5	Using an Existing Concrete WSDL .....	68
<b>7</b>	<b>Create a Technical Service Provider as an Application .....</b>	<b>70</b>
7.1	Template Overview .....	70
7.2	Pre-Requisites .....	70
7.2.1	Create the Application Module .....	70
7.2.2	Change the Module Properties .....	73
7.2.3	Create the Process Packages .....	75
7.2.4	Change the Service Process .....	76
7.2.5	Fix the Process Mappings.....	80
7.2.6	Back End on HTTP/HTTPS .....	87
7.2.7	Back End on JDBC .....	95
7.3	Implement the Service .....	96
7.3.1	Service Response.....	96
7.3.2	Invoking another ESB service.....	96
7.3.3	Validate the Request.....	96
7.3.4	Module Properties, Processes, Service Invocation.....	96
7.3.5	Using JMS Queue.....	96
7.3.6	Create the SOAP Binding .....	97
7.3.7	Create the EMS deployment script .....	106
7.3.8	Create the EMS undeployment scripts .....	107
<b>8</b>	<b>Setting up Automated Tests (SOAP over JMS) .....</b>	<b>108</b>
8.1	Pre-Requisites .....	108
8.2	Set up HermesJMS.....	108
<b>9</b>	<b>Create POM for Maven Build .....</b>	<b>113</b>
9.1	The Parent POM .....	113
9.2	The Application POM.....	113
9.3	The Module POM.....	114

## Table of Figures

---

Figure 1: Throw_ValidationFault sample .....	36
Figure 2: TS Service Provider Shared Module: Module Name .....	43
Figure 3: TS Service Provider Shared Module: Module Properties .....	44
Figure 4: TS Service Provider Shared Module: Resource Properties .....	46

Figure 5: TS Service Provider Shared Module: Process Name.....	47
Figure 6: TS Service Provider Shared Module: Implement Service Operation.....	48
Figure 7: TS Service Provider Shared Module: Change input transition .....	49
Figure 8: TS Service Provider Shared Module: Reply Transition .....	50
Figure 9: TS Service Provider Shared Module: AssignRH mapping .....	51
Figure 10: TS Service Provider Shared Module: Reply_Fault mapping .....	52
Figure 11: TS Service Provider Shared Module: Reply_BusinessFault mapping.....	53
Figure 12: TS Service Provider Shared Module: Reply_TechnicalFault mapping .....	54
Figure 13: TS Service Provider Shared Module: Reply_ValidationFault mapping .....	55
Figure 15: TS Service Provider Shared Module: Throw_TimeoutFault mapping .....	63
Figure 16: TS Service Provider Shared Module: Throw_TimeoutFault configuration....	64
Figure 16: TS Service Provider Shared Module: Module Name .....	72
Figure 17: TS Service Provider Shared Module: Module Properties .....	73
Figure 18: TS Service Provider Shared Module: Resource Properties .....	75
Figure 19: TS Service Provider Shared Module: Process Name.....	76
Figure 20: TS Service Provider Shared Module: Implement Service Operation.....	77
Figure 21: TS Service Provider Shared Module: Change input transition .....	78
Figure 22: TS Service Provider Shared Module: Reply Transition .....	79
Figure 23: TS Service Provider Shared Module: AssignRH mapping .....	80
Figure 24: TS Service Provider Shared Module: Reply_Fault mapping .....	81
Figure 25: TS Service Provider Shared Module: Reply_BusinessFault mapping.....	82
Figure 26: TS Service Provider Shared Module: Reply_TechnicalFault mapping .....	83
Figure 27: TS Service Provider Shared Module: Reply_ValidationFault mapping .....	84
Figure 28: TS Service Provider Shared Module: Throw_TimeoutFault mapping .....	92
Figure 29: TS Service Provider Shared Module: Throw_TimeoutFault configuration....	93
Figure 17: Create the SOAP over JMS Interface: Module Components .....	97
Figure 18: Create the SOAP over JMS Interface: Service Settings.....	98
Figure 19: Create the SOAP over JMS Interface: New SOAP Binding .....	99
Figure 20: Create the SOAP over JMS Interface: New SOAP Binding .....	100
Figure 21: Create the SOAP over JMS Interface: Transport Type .....	101
Figure 22: Create the SOAP over JMS Interface: JMS Settings .....	102
Figure 23: Create the SOAP over JMS Interface: Queue Name as Property .....	103

## Table of Tables

---

Table 1: Acronyms .....	9
Table 2: Issues .....	10

## 1 Document Control

### 1.1 Versioning Information

Issue Date	Version	Description	Author
06/01/2016	0.1.0	First draft – Based on reference document from Pierre Ayel	Julian Cranfield

### 1.2 Distribution Information

#### 1.2.1 CO-OP Bank

Name	Project Role

#### 1.2.2 TIBCO

Name	Project Role
Andrew Dearing	Project Delivery Manager

### 1.3 Reference Documents

Name	Author

## 1.4 Document Acceptance

Signed for and on behalf of The Automobile Association	
Signature of authorised officer:	
Name of authorised officer:	
Witness:	
Date:	

## 2 Introduction

To enable the standardisation of services within CO-OP Bank using TIBCO BusinessWorks 6, a few templates have been created as starting points for the creation of services.

These templates do not dictate who the service should operate or the orchestration of the activities within the service process, however, they do position audit, exception handing within the process orchestration in a standardised way.

The naming conventions in this document have been created to ensure a standard across all services and therefore a general understanding. For example, when trying to find the endpoint URL for a service in the module properties there will be a standard naming convention so this can be located quickly and accurately.

### 2.1 Scope

This document and the templates that it references enable the creation of business and technical services.

### 2.2 Purpose

The templates and the use of the templates as laid out in this document ensure the standardization of TIBCO BusinessWorks development for services.

### 2.3 Audience

All developers of services using TIBCO BusinessWorks should adhere to the practices in this document.

### 2.4 Acronyms

*Table 1: Acronyms*

Acronym	Definition
BST	TIBCO Business Studio: The Eclipse IDE for building BW applications
BW	TIBCO BusinessWorks
EMS	TIBCO Enterprise Messaging Service
ESB	Enterprise Service Bus, a technical capability providing runtime facilities for transformation and mapping, integration and process management (orchestration)
FTP	File Transfer Protocol
JDBC	Java API: generic API to integrate with databases
JMS	Java API: Java Messaging Service
JSON	Data format: Javascript Object Notation
HTTP	Network transport protocol: Hypertext Transport Protocol

<b>Acronym</b>	<b>Definition</b>
GIT	Git source control system
XML	Data format: Extended Markup Language

## 2.5 Issues

Table 2: Issues

<b>#</b>	<b>Status</b>	<b>Owner</b>	<b>Description / Resolution</b>

## 2.6 Risks

TBC

### 3 Introduction to the Templates

**Business service** – these are front-end requests that users see. Business services call to technical service(s) for action.

**Technical service** – these provide the undercover operations, like talking to a database, and they are unaware of the business services and business logic.

**Application** – applications are **deployable** which means there is either a SOAP service bound to JMS or HTTP, or REST over HTTP.

**Shared module** – these are **not deployable** since they have no bindings, and are used for inline calls only.

## 4 Create a Business Service Provider as an Application

This chapter explains how to create a new Business service exposed on SOAP over HTTP/JMS.

### 4.1 Template Overview

The template module APP\_BSServiceTemplate.module allows you to create a new application for a Business service. It contains the following objects:

- Processes:
  - **BSService**: the service implementation.
  - **Activator**: the process executed when the application starts up inside an appnode. This can be used to trace configuration settings (such as back-end URLs) or call other Activator required in dependent modules.
- WSDLs:
  - **ModuleActivator.wsdl**: the abstract WSDL of the **Activator** process (this one should never be changed at all: it is created by BW when the Activator process is created).
- Resources:
  - **Server-BW01**: the HTTP server where the Business service will be exposed.
  - **JNDIClient-ESB01**: the JNDI connection to the ESB Server that the Business service MUST use when invoking other ESB services exposed on SOAP over JMS.

**Notes:**

- 1 The JNDI and JMS connections used by the System Error Handler and System Audit are separated from JNDIClient-ESB01 and JMSClient-ESB01. They are contained in the LIB\_TSUTIL\_SystemErrorHandler\_Client and LIB\_TSUTIL\_SystemAudit\_Client modules.

### 4.2 Pre-Requisites

The XSD and WSDL have been created with all the correct naming structures.

#### 4.2.1 Create the Application Module

With Windows Explorer:

1. Copy the Service Application Module template folder APP\_BSServiceTemplate.module into the folder **<GIT> / trunk / BW / BusinessServices**.
2. Rename the copied folder into “APP\_BS<Area>\_<ServiceName>”.
3. With a text editor, open the project file into the copied folder and change the project name at the top from APP\_ServiceTemplate.module into APP\_BS<Area>\_<ServiceName>.

**Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>APP_BSGEN_ServiceA</name>
    <comment></comment>
    ...

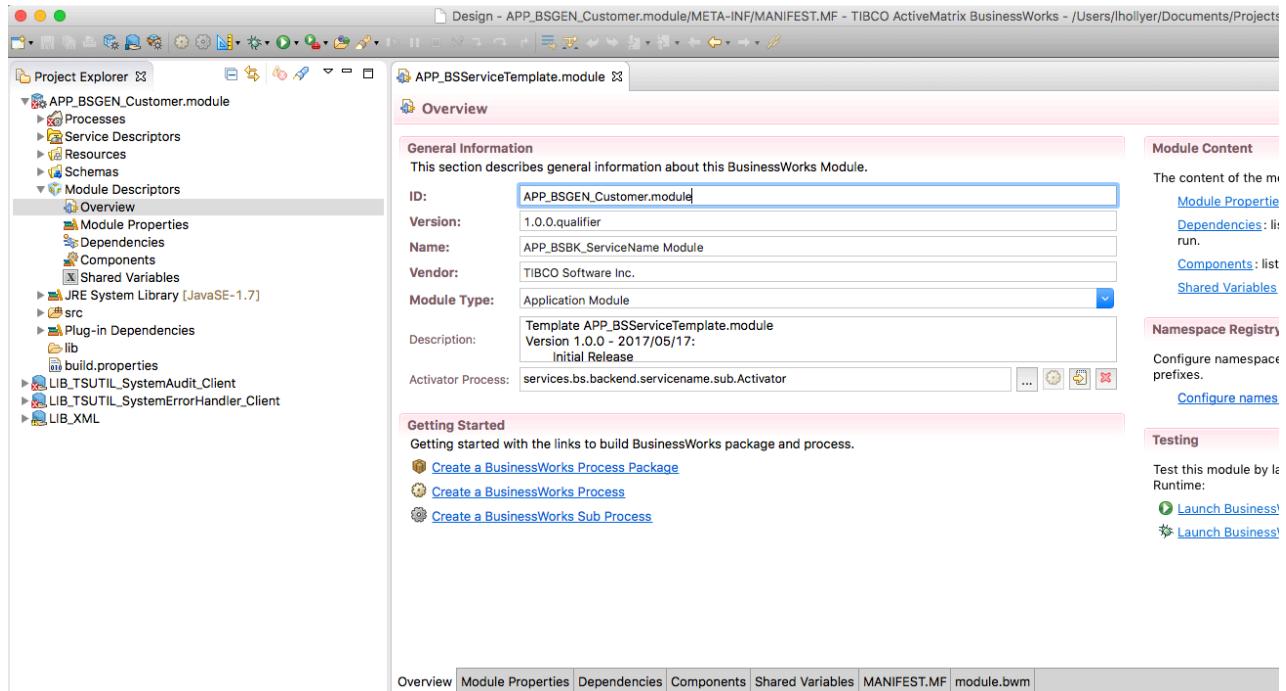
```

With TIBCO BusinessStudio

4. Start TIBCO BusinessStudio
5. In your workspace, import the following modules:
  - a. LIB\_XML (from <GIT> / trunk / XML)
  - b. LIB\_TSUTIL\_SystemAudit\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - c. LIB\_TSUTIL\_SystemErrorHandler\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - d. APP\_<TYPE><Area>\_<ServiceName> (from <GIT> / trunk / BW / BusinessServices)

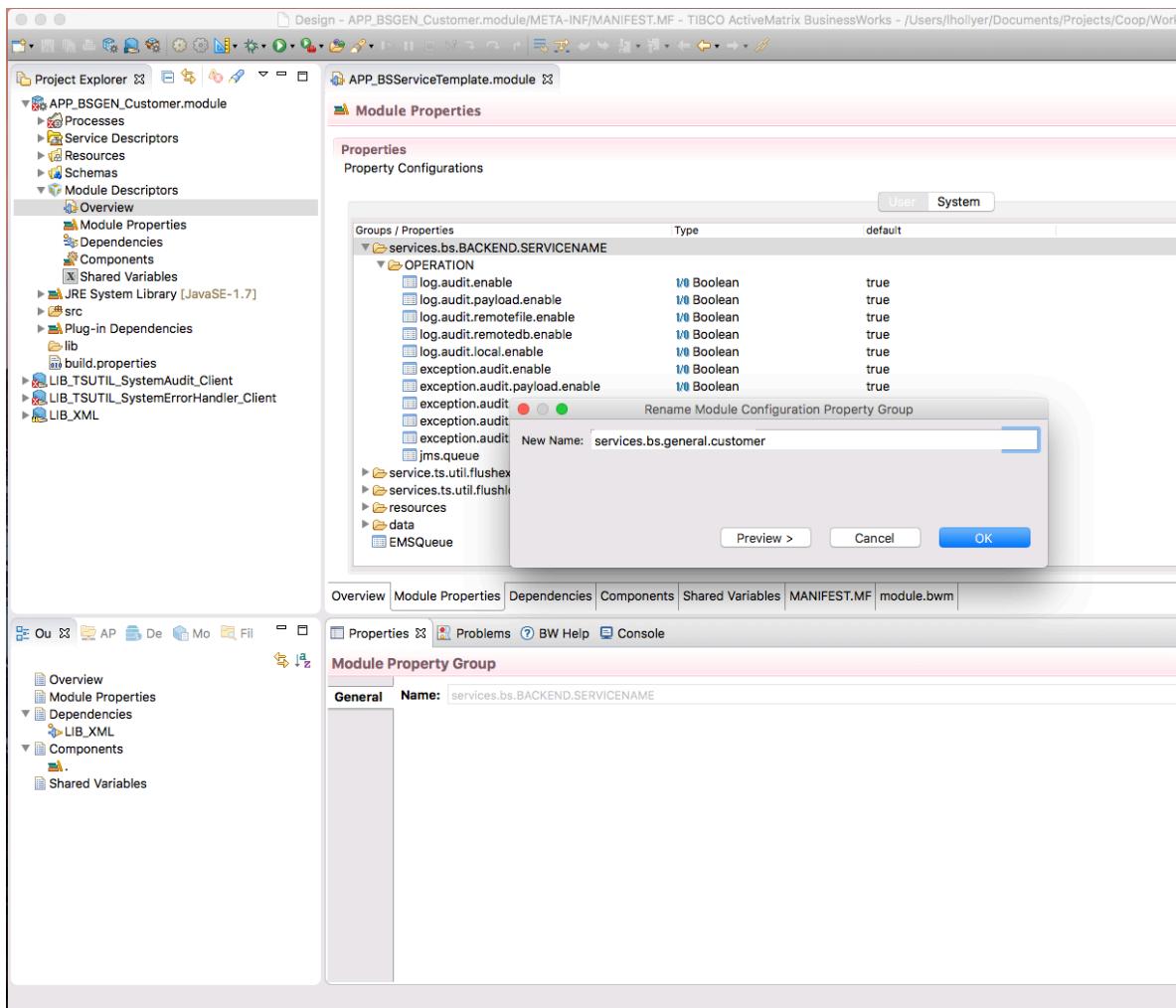
All subsequent changes are done on the APP\_<TYPE><Area>\_<ServiceName> module:

6. Go to the module overview.
7. Change the name into “APP\_BS<Area>\_<ServiceName> Module”:

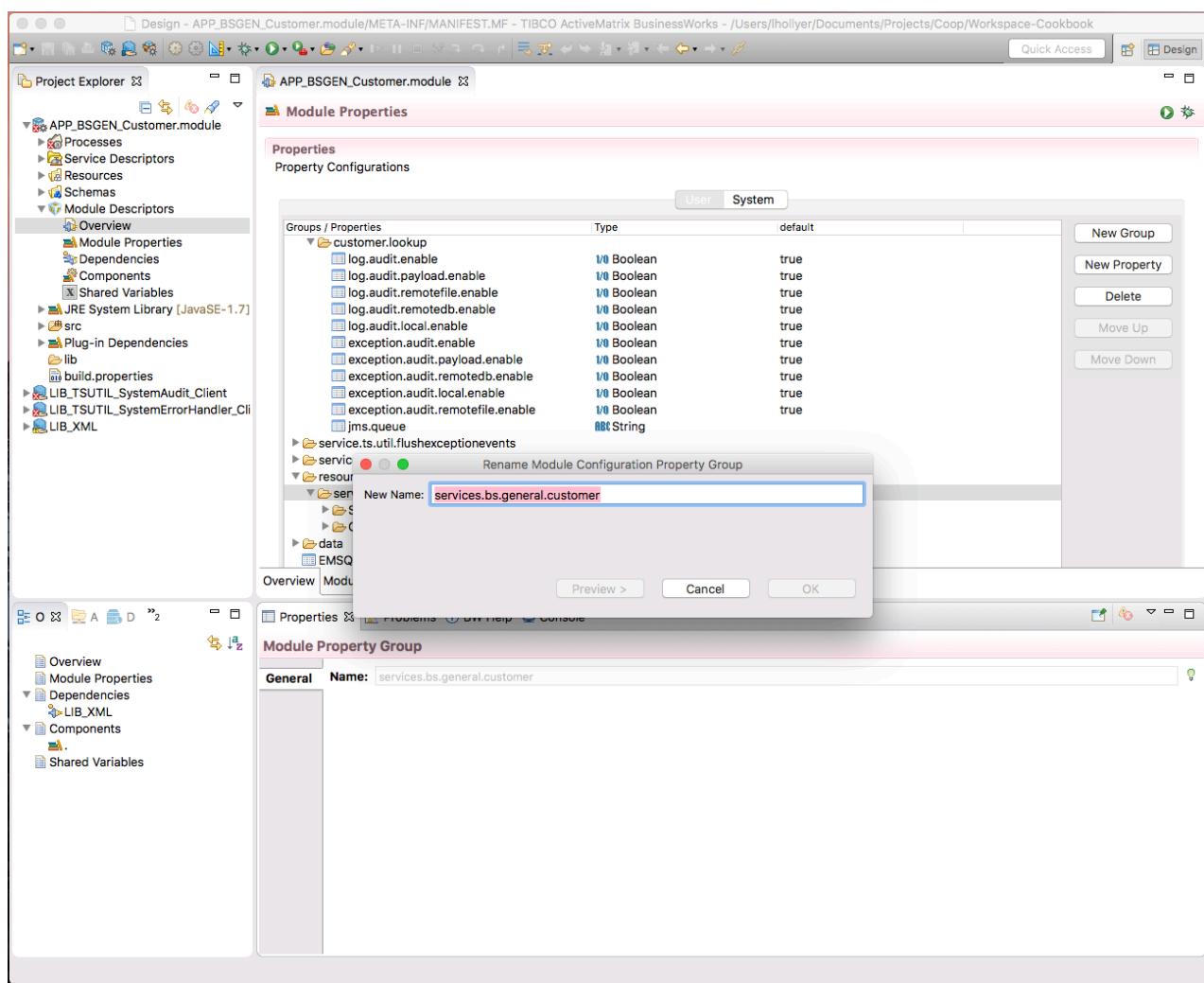


#### 4.2.2 Change the Module Properties

1. Go to the module properties.
2. Select the group “services.bs.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.<type><area>.<servicename>”:



3. In this group, use the same technique to rename the group “OPERATION” into “<ServiceName>”.
4. Select the group “resources / services.bs.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.<type><area>.<servicename>”:



- In this group, change the value of the HTTP port to the port assigned for the service.

#### 4.2.3 Rename the Resources

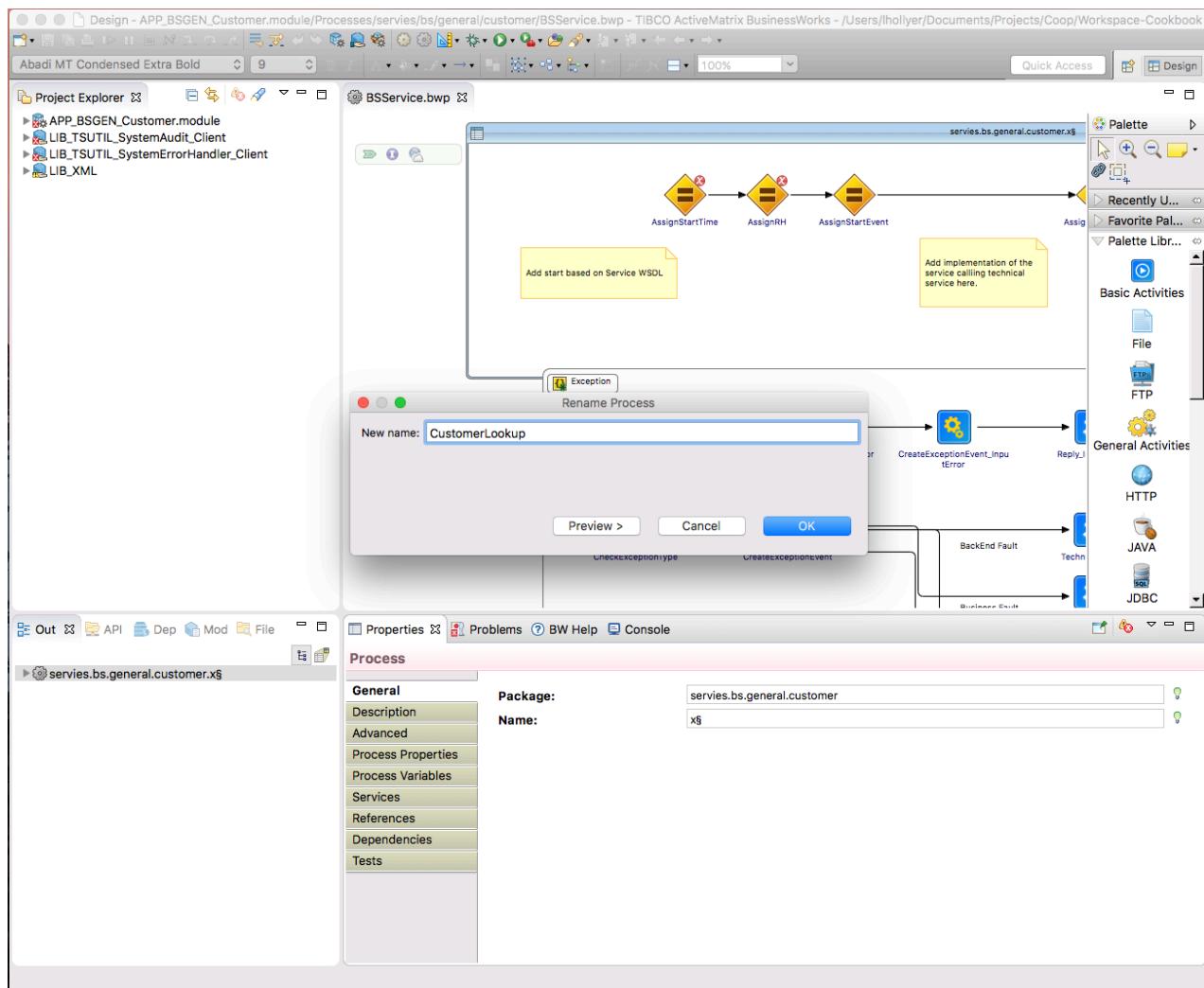
- Open the HTTP Server resource. In the properties pane, use the light-bulb icon to change the name into "services.<type>.<area>.<servicename>.Server-BW01" (only the package name should change).
- Do the same for the JNDIClient-ESB01 resource.

#### 4.2.4 Create the Process Packages

- Create the process package "services.<type>.<area>.<servicename>".
- Move the BSService process into it.
- Create the process package "services.<type>.<area>.<servicename>.sub".
- Move the Activator process into it.
- Delete the remaining ...ServiceName... process packages.

#### 4.2.5 Change the Service Process

1. Open the process “services.<type>.<area>.<servicename> / BSService”.
2. In the properties, use the light-bulb icon to rename it into “<ServiceName>”:

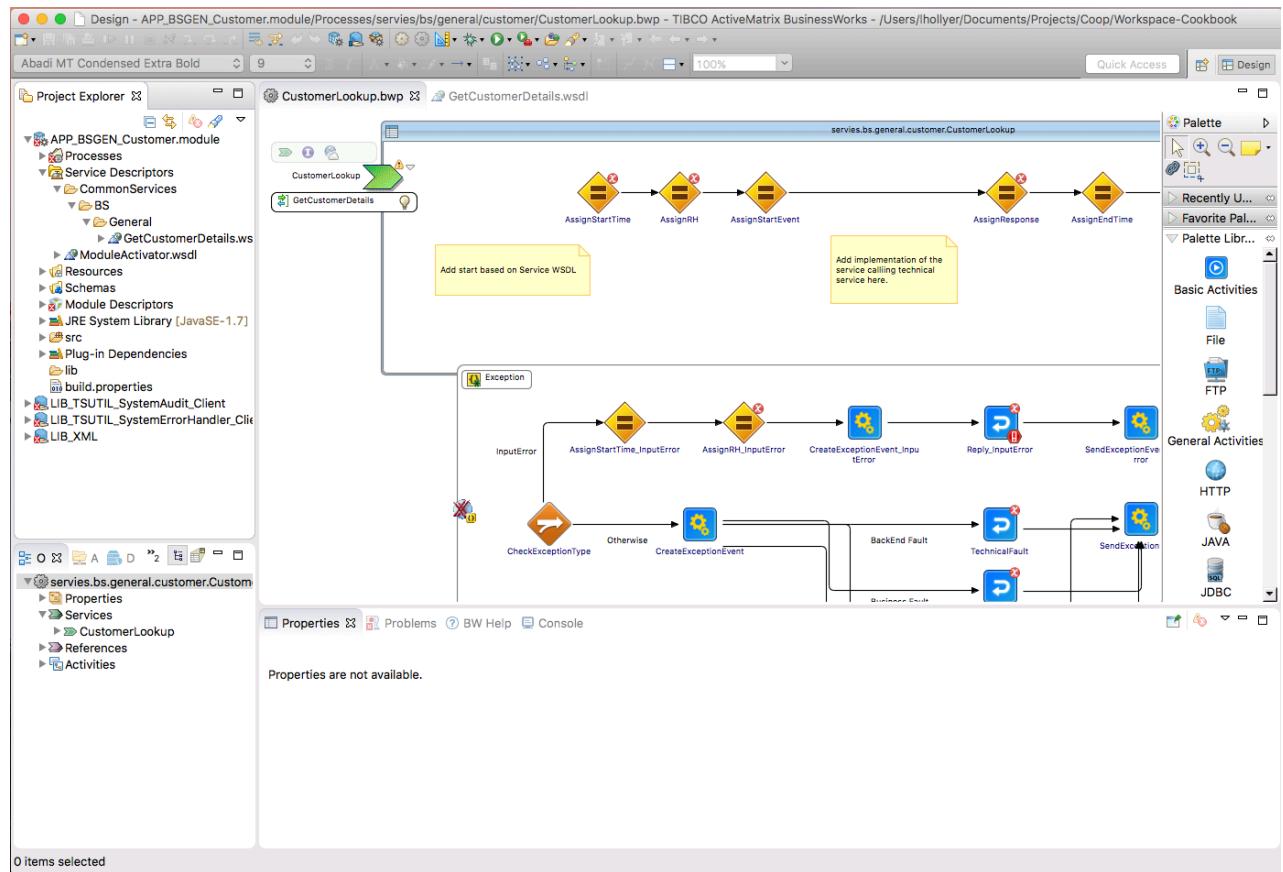


3. Import the XSD and WSDL into the module.

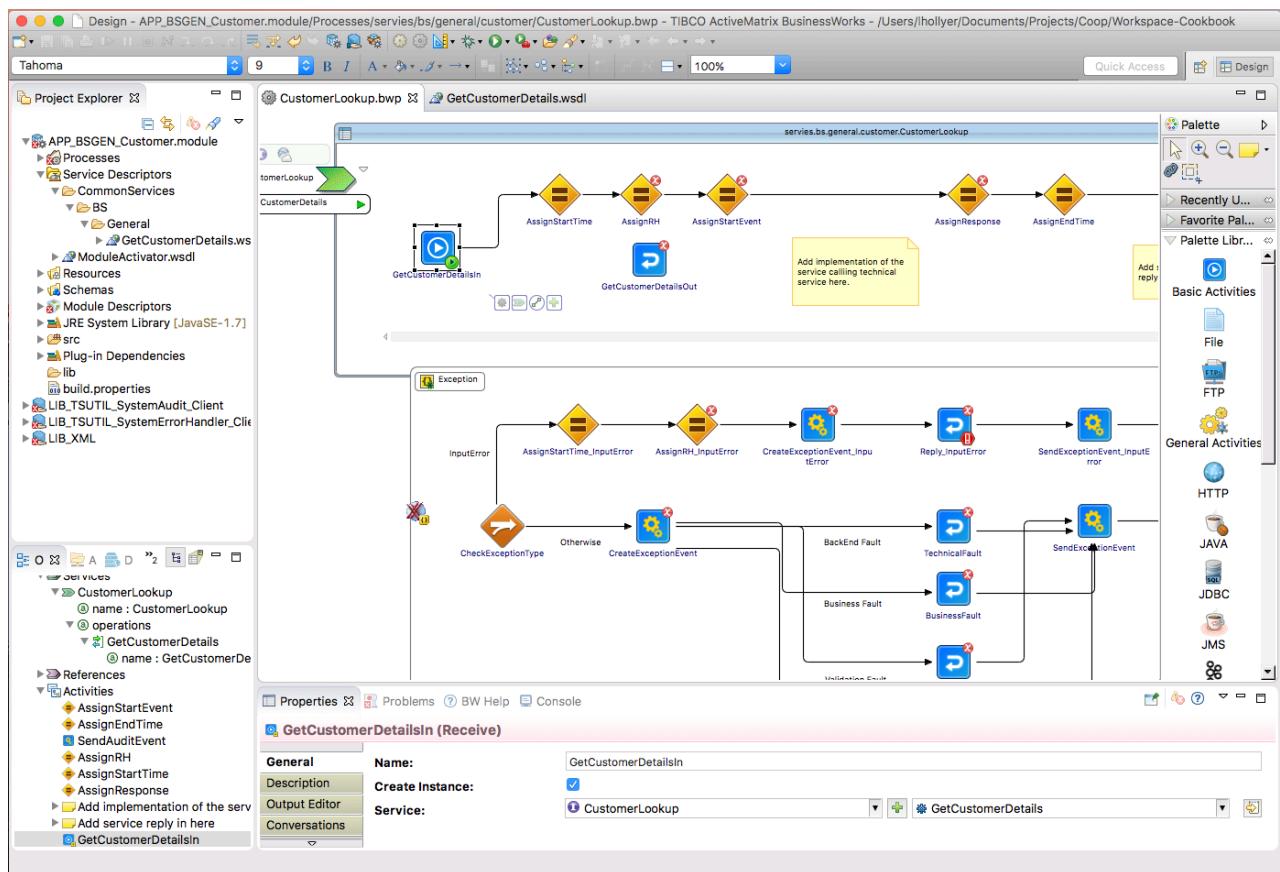
- a. Follow the **ServiceName.xsd** example in **Schemas / CommonServices / ... / ServiceName** and make sure the targetNamespace and tns match up in the file directory, and the simple and complex types are renamed.
- b. Ensure **ServiceName.wsdl** located in **Service Descriptors / CommonServices / ... / ServiceName** has message names and inputs/outputs matching up with **ServiceName.xsd**.

In case they do not match up, change the imports in the source code in **ServiceName.wsdl** to match the file structure.

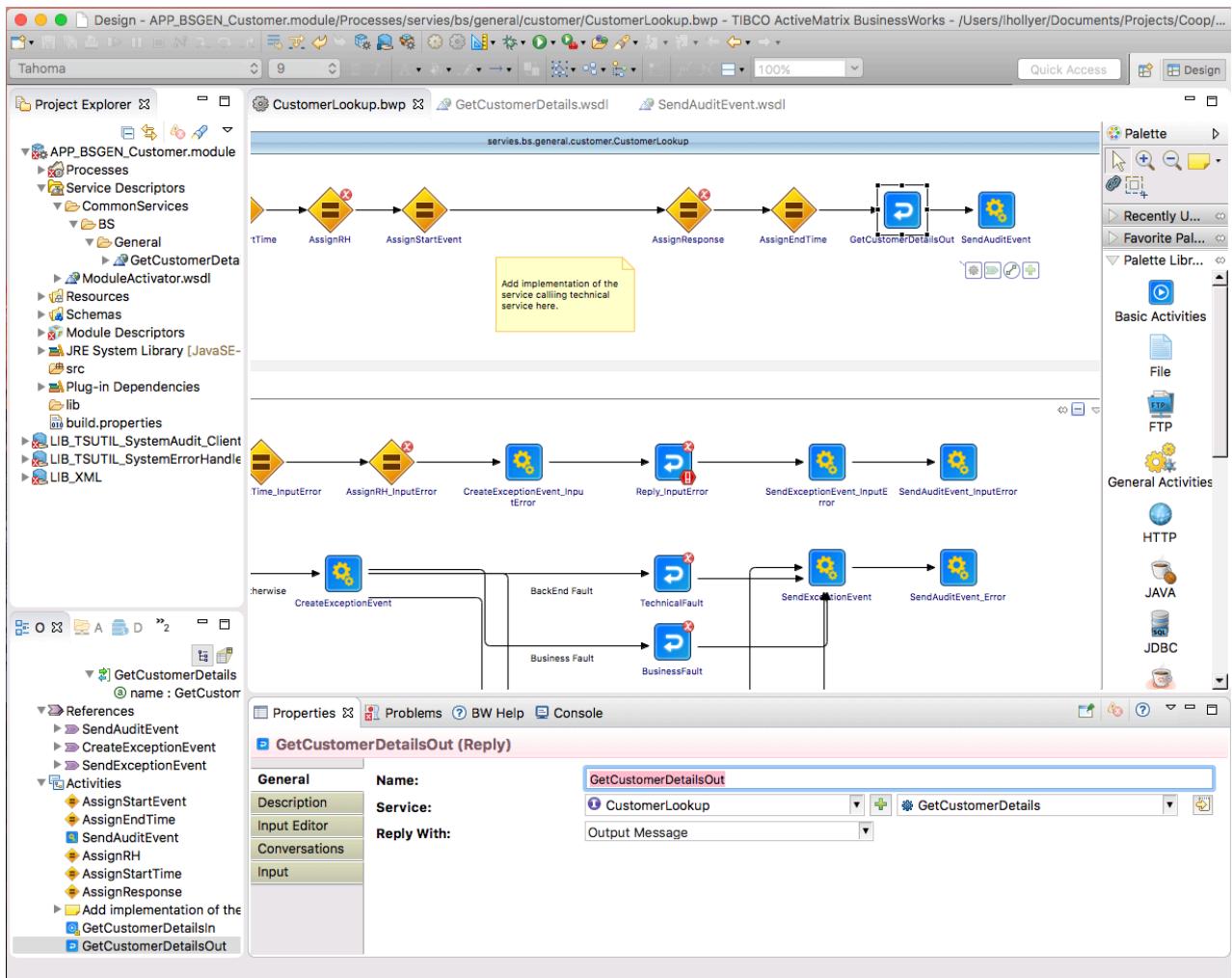
4. Drag and drop the abstract WSDL into the process so the service operation is implemented:



5. Create a transition between the <Operation>In task and “AssignStartTime” task:

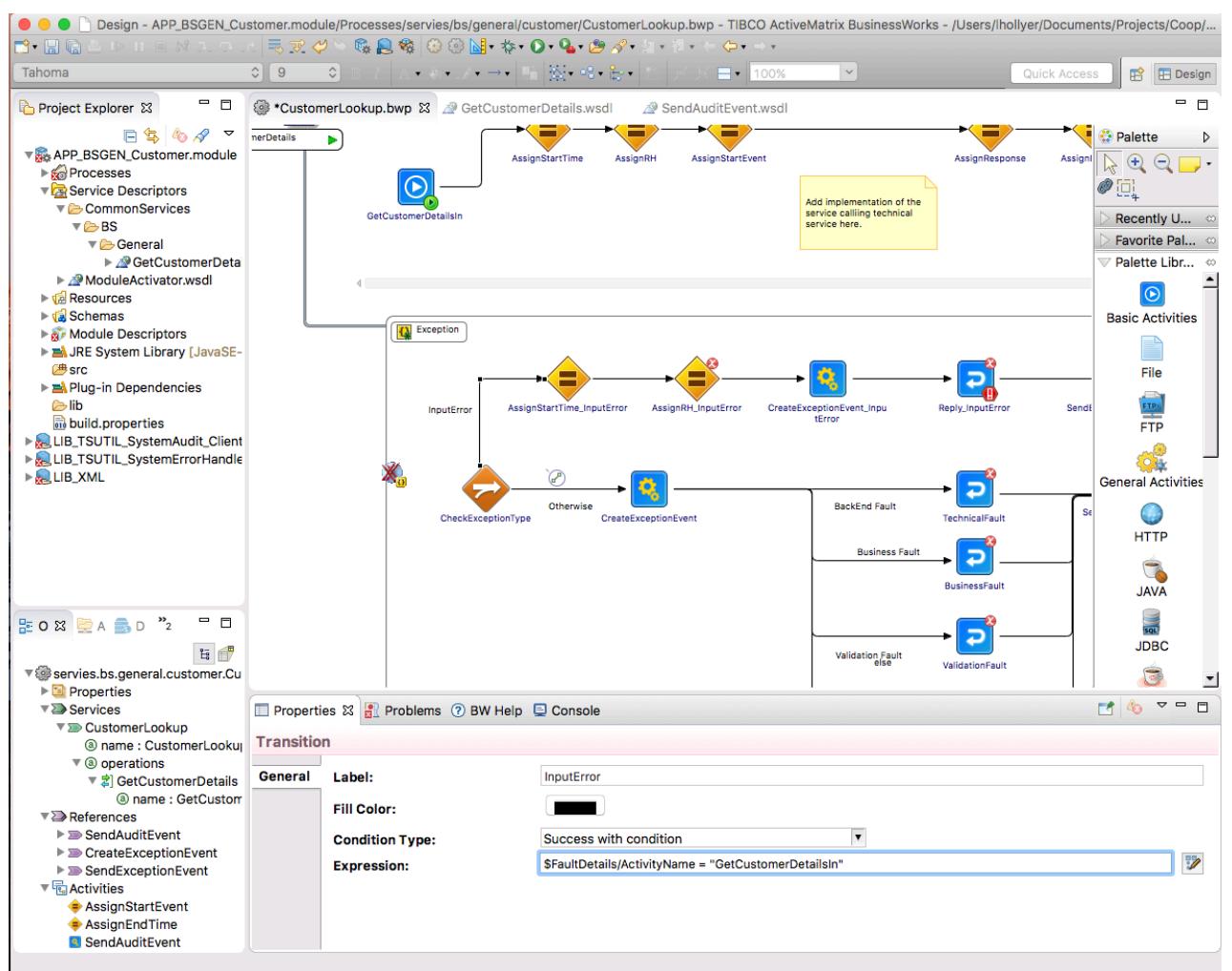


6. Re-arrange transitions so the <Operation>Out task is located between “AssignEndTime” and “SendAuditEvent” tasks:



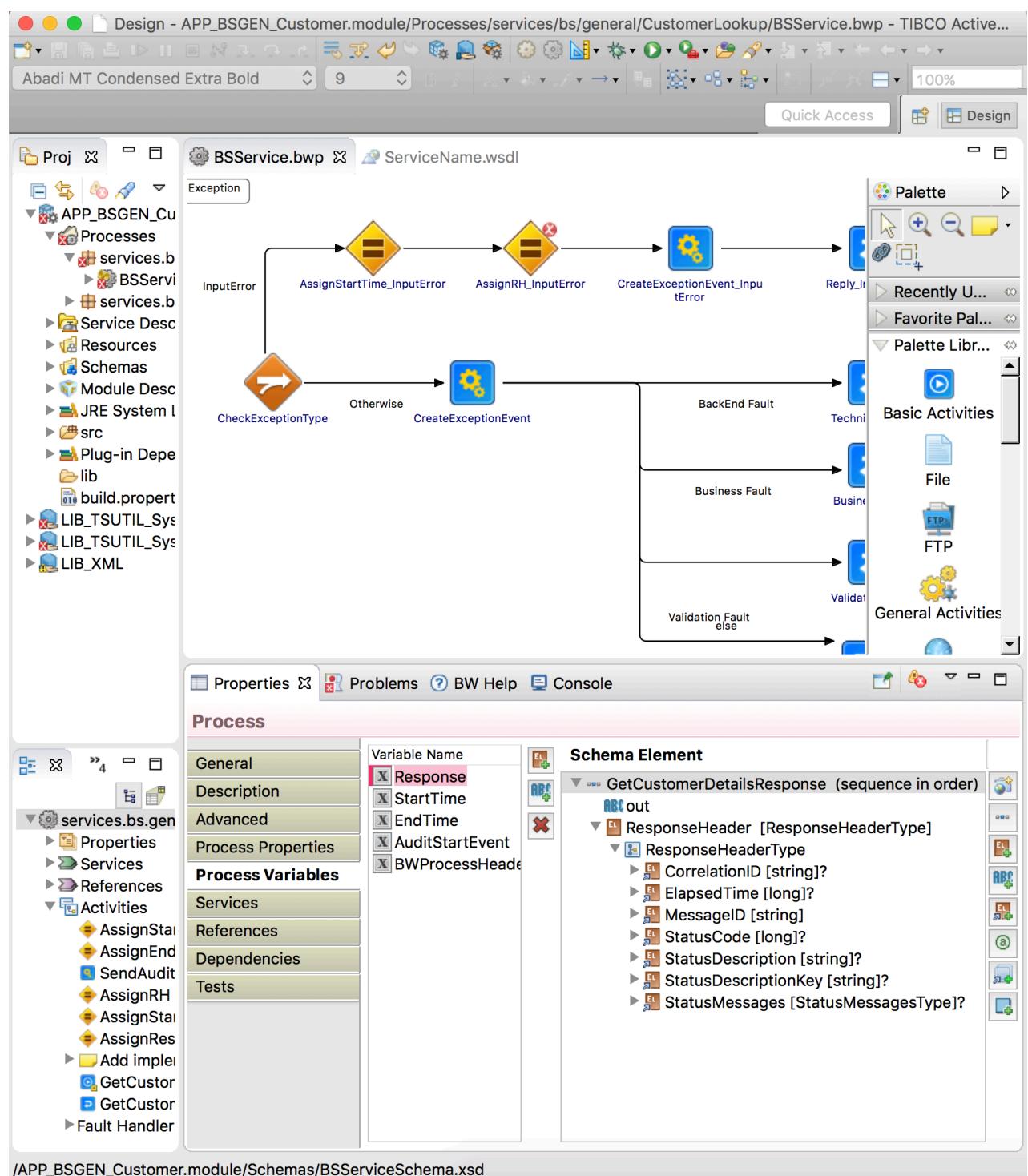
7. Fix the condition in the transition named “InputError” in the catch-all block as following:  
 $\$FaultDetails/ActivityName = "<\text{Operation}>\text{In}"$ .

## CO-OP Bank - ESB – How To: Create BW Services from Templates



8. Go the Process Variables. Change the schema of the “Response” variable so it uses the “<Operation>Response” element from the service XSD:

If the response header doesn't change automatically, add a new element called “ResponseHeader” and select the schema to be “ResponseHeaderType”. The child elements should automatically be added.



9. Save your changes.

## 4.2.6 Fix the Process Mappings

### 4.2.6.1 AssignRH

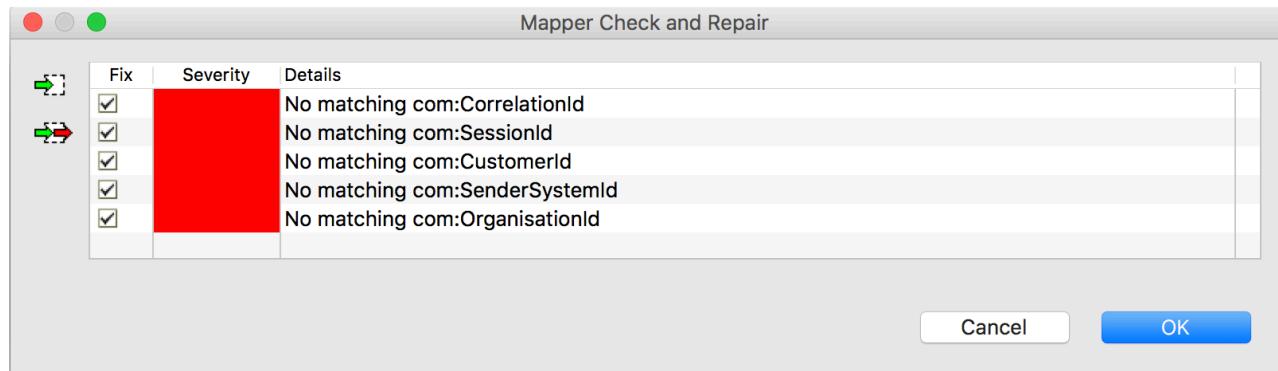
This task prepares the message header that can be passed into calls to other ESB services or returned in the reply or fault messages.

Fix the mapping in the task “AssignRH” as following:

1. Change the formula for the “varRequest” variable into

```
$<Operation>In/parameters/tns9:<Operation>Request
```

2. Fix the mapping of the CorrelationId in the When statement, if necessary (use the fixing icon from Eclipse):



Cancel

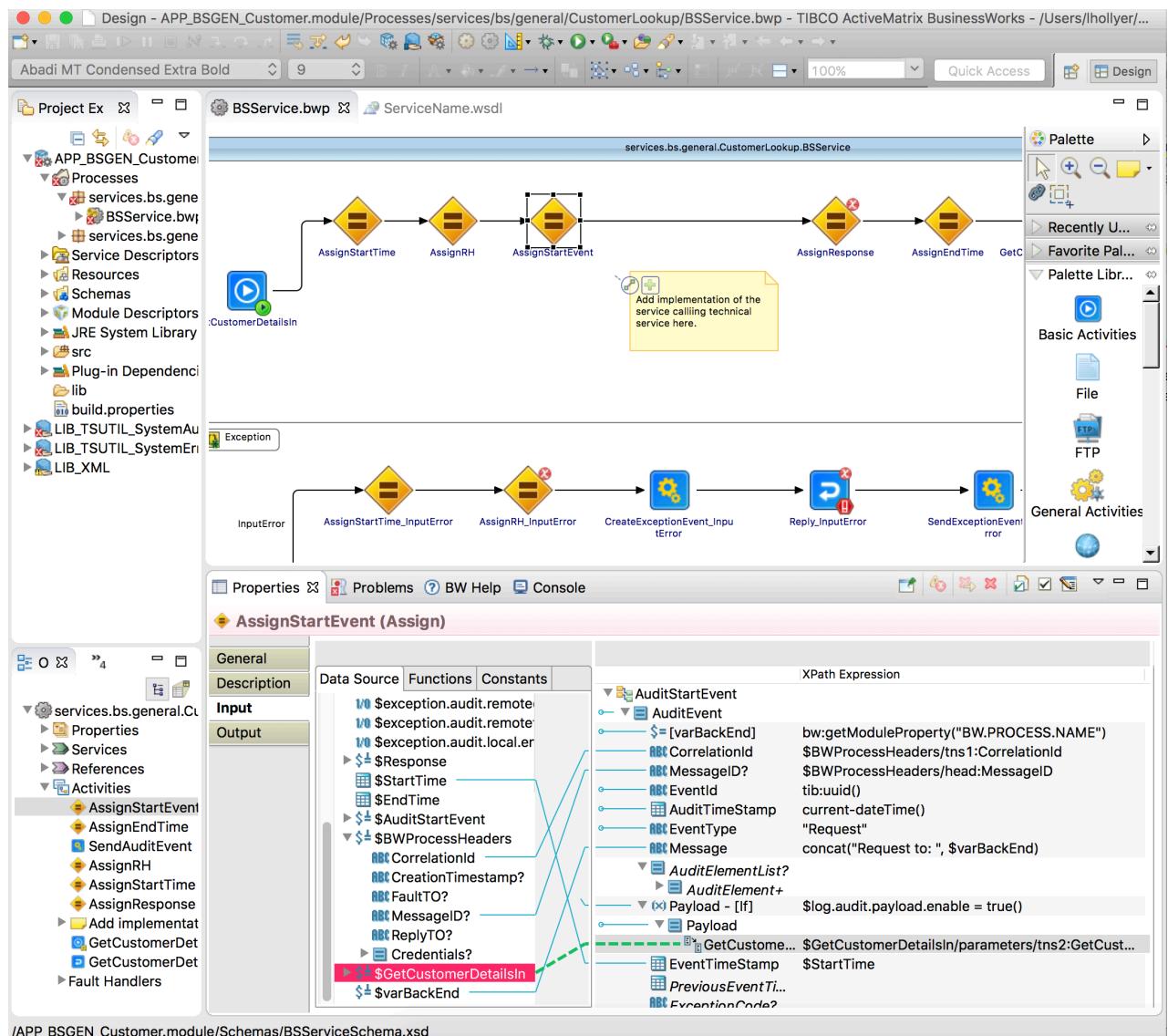
OK

#### 4.2.6.2 AssignStartEvent

This task prepares the Audit Event indicating this service has received a request from a consumer. This event will be passed into the SendAuditEvent and SendAuditEvent\_Error tasks.

Fix the mapping in “AssignStartEvent” as following:

1. Payload content should be a copy of the <Operation>In/parameters/<Operation>Request XML element. The “Payload” element must have a surrounding “If” statement which formula is “\$log.audit.payload.enable = true()”:

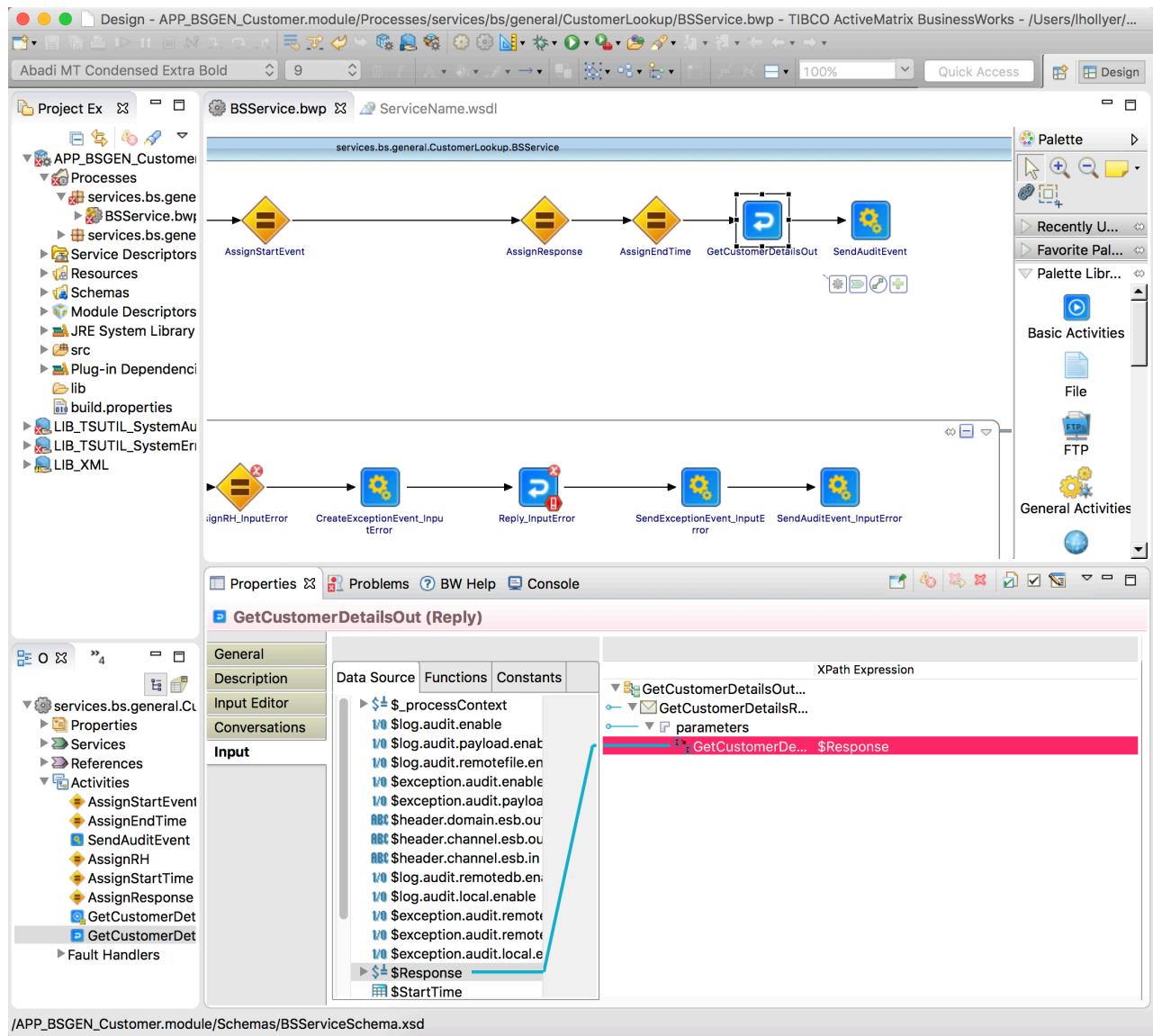


#### 4.2.6.3 <Operation>Out

This task returns a response to the consumer in case of success.

Fix the mapping as following:

1. AssignResponse – map “out” and “ResponseHeader from \$Response.
2. <Operation>Response/parameters/<Operation>Response should be a copy-of the \$Response XML element:



#### 4.2.6.4 Reply\_Error

This tasks returns a ValidationFault SOAP fault to the consumer.

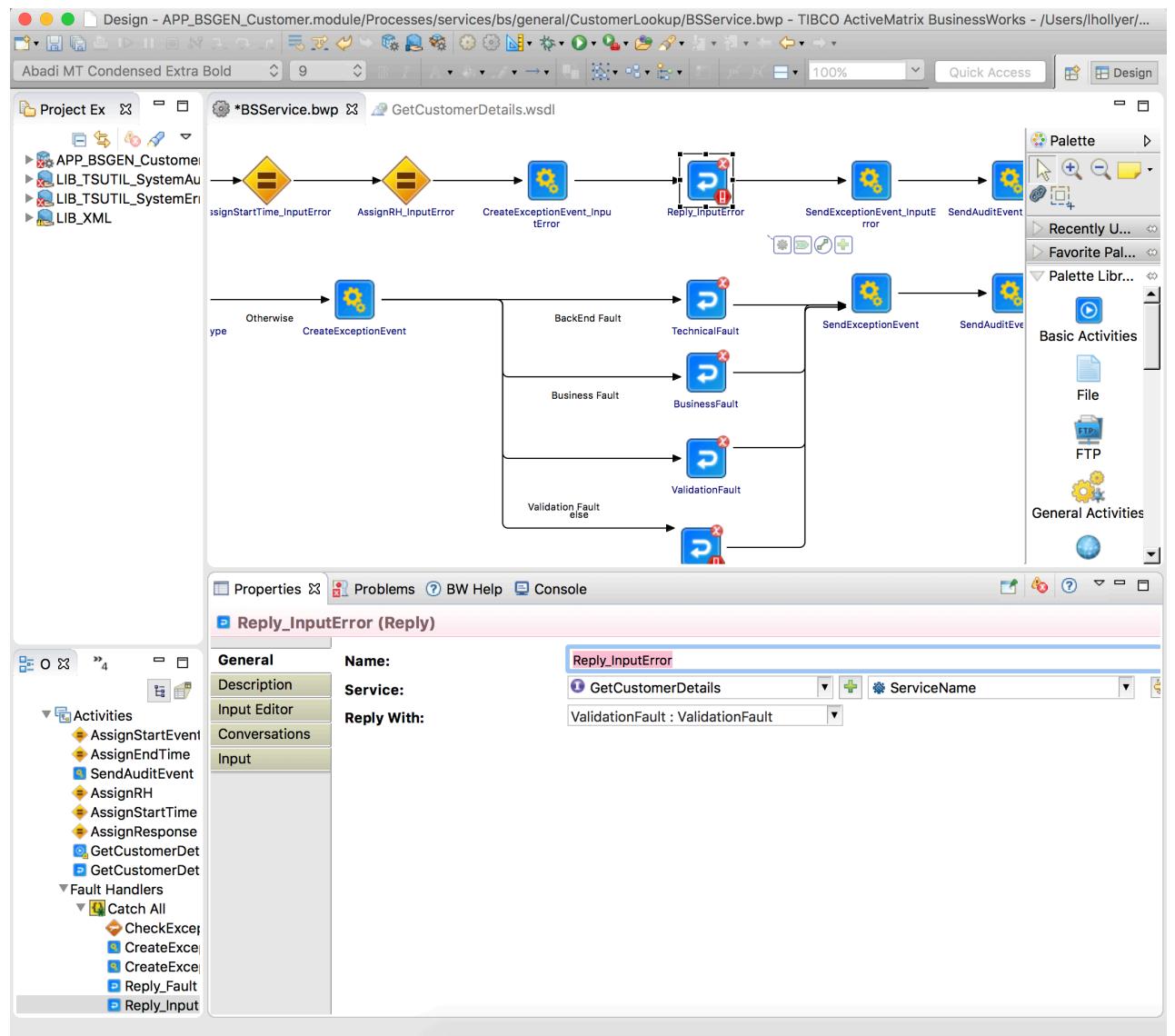
1. Change AssignRH\_Error/varRequest to:  
\$GetCustomerDetailsIn/parameters/tns1:GetCustomerDetailsRequest

Fix the configuration of the Reply\_InputError task (select the service, operation and “ValidationFault”). **If a concrete WSDL is used, change ‘Reply With’ to ‘Undeclared Fault’ instead.**

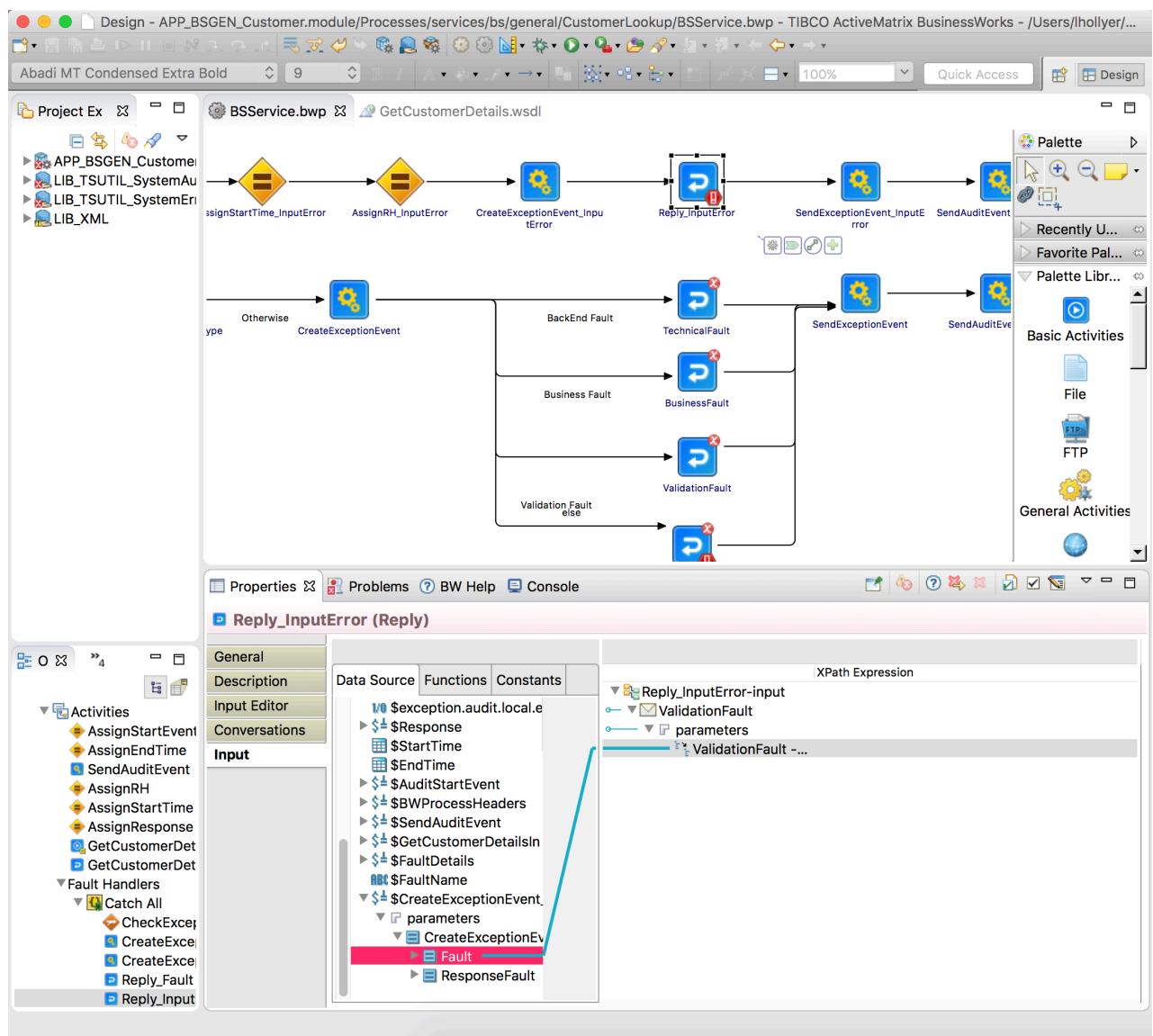
1. Fix the mapping of the Reply\_InputError task so ValidationFault element is a copy of the CreateExceptionEvent\_InputError / Response / ValidationFault element.

If ValidationFault element doesn't appear, check on the General tab the ‘Service’ is <ServiceName> and ‘Reply With’ is set to ‘ValidationFault’

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity.



## CO-OP Bank - ESB – How To: Create BW Services from Templates



#### 4.2.6.5 *Reply\_Fault*

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML Fault element.

Fix the configuration of the Reply\_Fault task:

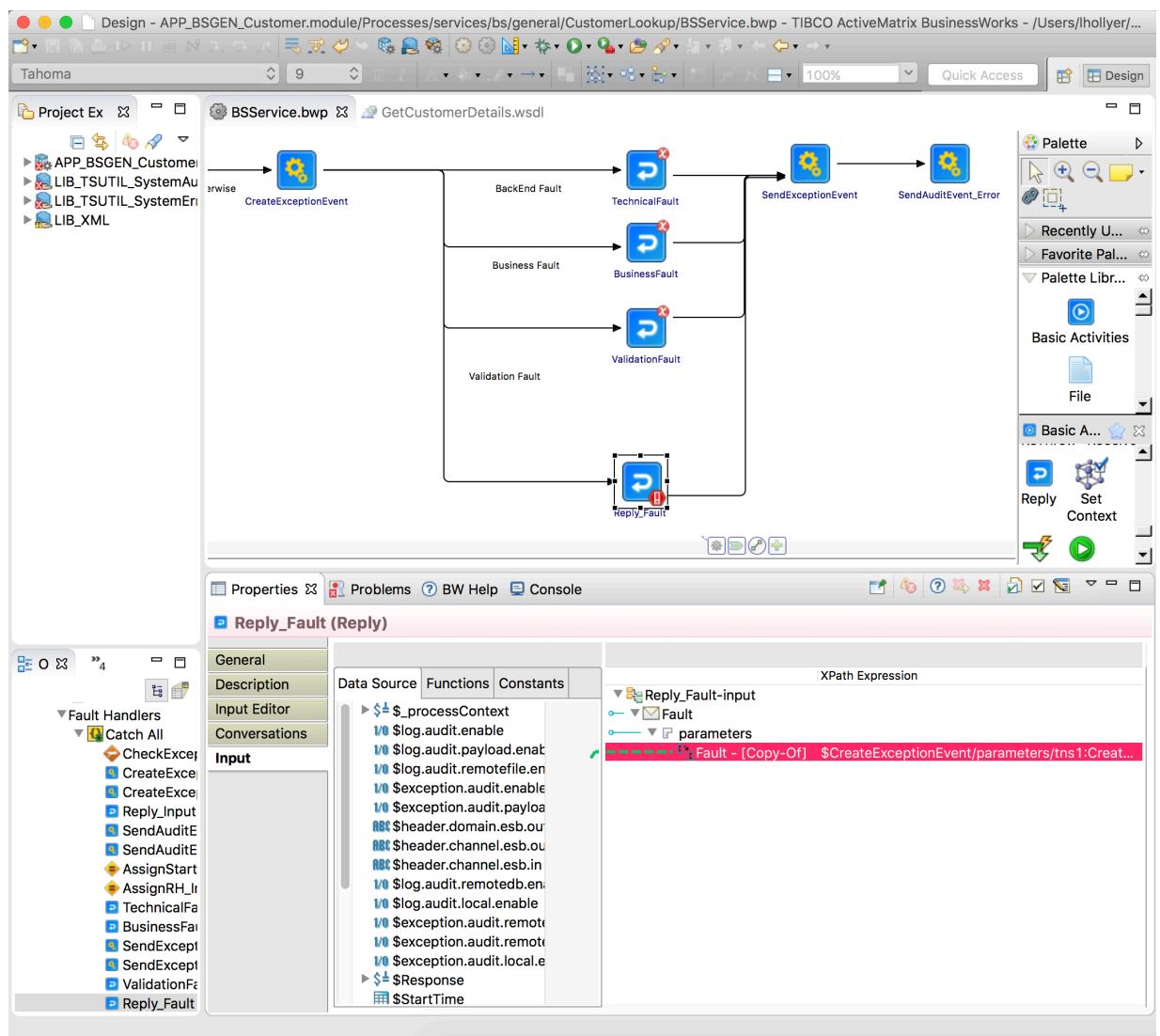
1. Select the service, operation and “Fault”. **If a concrete WSDL is used, change ‘Reply With’ to ‘Undeclared Fault’ instead.**

Fix the mapping of the Reply\_Fault task so.

Fault element is a copy of the CreateExceptionEvent / Response / Fault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with no matching condition”.

## CO-OP Bank - ESB – How To: Create BW Services from Templates



#### **4.2.6.6 *TechnicalFault***

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML TechnicalFault element.

Fix the configuration of the TechnicalFault task:

1. Select the service, operation and “TechnicalFault”. **If a concrete WSDL is used, change ‘Reply With’ to ‘Undeclared Fault’ instead.**

Fix the mapping of the TechnicalFault task so:

2. TechnicalFault element is a copy of the CreateExceptionEvent / Response / TechnicalFault element.

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” - exists(\$CreateExceptionEvent/parameters/tns3>CreateExceptionEventResponse/tns3:ResponseFault/tns2:TechnicalFault).

#### **4.2.6.7 *BusinessFault***

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML BusinessFault element.

Fix the configuration of the BusinessFault task:

1. Select the service, operation and “BackEndFault”. **If a concrete WSDL is used, change ‘Reply With’ to ‘Undeclared Fault’ instead.**

Fix the mapping of the BusinessFault task so:

2. BusinessFault element is a copy of the CreateExceptionEvent / Response / BusinessFault element.

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” - exists(\$CreateExceptionEvent/parameters/tns3>CreateExceptionEventResponse/tns3:ResponseFault/tns2:BusinessFault).

#### **4.2.6.8 *ValidationFault***

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML ValidationFault element.

Fix the configuration of the ValidationFault task:

1. Select the service, operation and “ValidationFault”. **If a concrete WSDL is used, change ‘Reply With’ to ‘Undeclared Fault’ instead.**

Fix the mapping of the ValidationFault task so:

2. ValidationFault element is a copy of the CreateExceptionEvent / Response / ValidationFault element.

**N.B. If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” –**

**exists(\$CreateExceptionEvent/parameters/tns3:CreateExceptionEventResponse/tns3:ResponseFault/tns2:ValidationFault).**

#### 4.2.7 Create the SOAP Binding

1. Create the SOAP Binding by clicking on “Create Service”
2. Drop and drag the newly created operation into the process.

### 4.3 Implement the Service

#### 4.3.1 Business Logic

1. Implement the Service: put the logic task between the “AssignStartEvent” and “AssignResponse” tasks.

#### 4.3.2 AssignResponse

1. The task AssignResponse maps the business logic data into the reply to be sent by the service.

You MUST map the ResponseHeader element as following:

- a. CorrelationId: \$BWProcessHeaders / CorrelationId
- b. MessageId: \$BWProcessHeaders / MessageId
- c. Version: \$BWProcessHeaders / Response / Version
- d. ResponseDateTime: current-dateTime()

#### 4.3.3 Error Handling

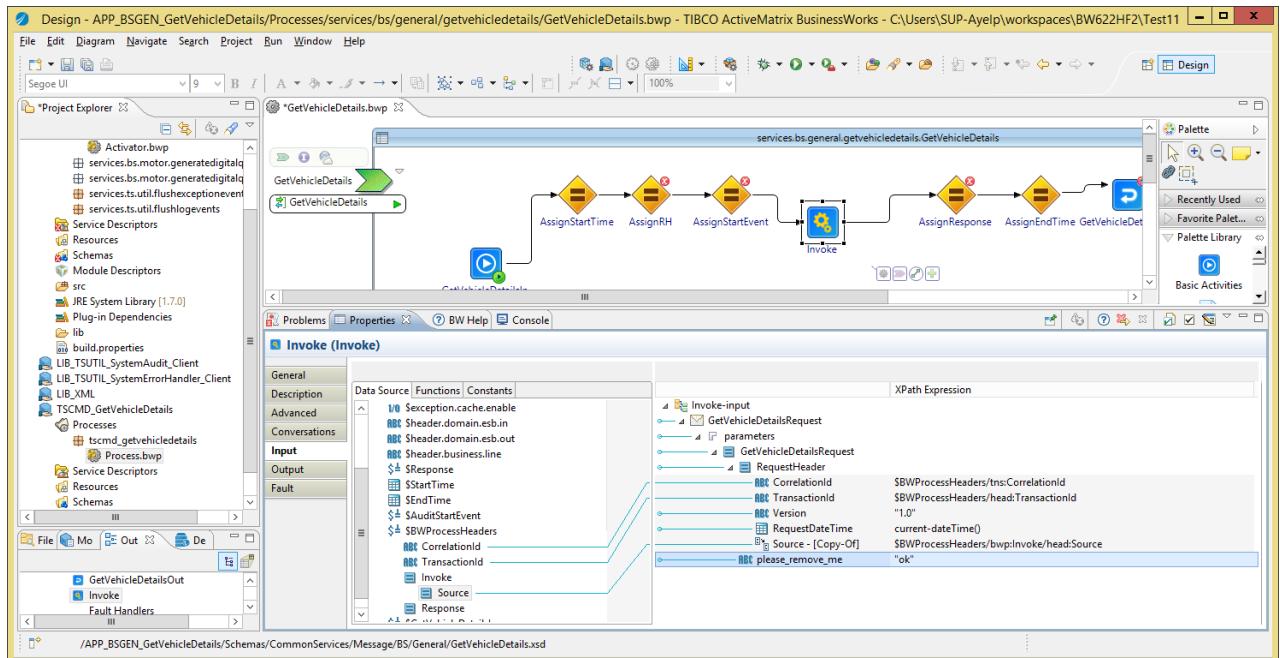
1. Depending on the design, some services may return a valid response in case of fault. In such case:
  - a. edit the mapping AssignErrorResponse task
  - b. Change the configuration of the Reply\_Fault task so it return a response and the mapping use the \$Response process variable.
  - c. You can do the same for other branches of fault if certain faults must return a valid reply and not a fault:
    - i. Add an Assign task to assign values into the \$Response variable.

- ii. Change the configuration and mapping of the XXXFault task so it return the \$Response as valid response instead of a fault.

#### 4.3.4 Invoke another ESB service

1. If you invoke another ESB service (BS or TS), you MUST map the RequestHeader element as following:
  - a. CorrelationId: \$BWProcessHeaders / CorrelationId
  - b. MessageId: \$BWProcessHeaders / MessageId
  - c. Version: The version of the invoked service, for example “1.0”.
  - d. RequestDateTime: current-dateTime()

In this screenshot we invoke TSGEN\_GetVehicleDetails TS service:



#### 4.3.5 Validate the Request

Some service may require additional input data validation (besides what is defined inside the service schema, which BW validates by default). In such case, you must implement the validation as following:

For a single validation rule:

1. Add a Throw task to the process,
2. Rename the task into “Throw\_ValidationFault”
3. Create a transition from the “AssignStartEvent” task to the “Throw\_ValidationFault” task.
4. Change the transition condition type to “Success with Condition”.
5. In the “Expression” field, type the validation condition which will evaluate to “true” if the rule is broken.
6. Change the condition type of the transition between “AssignStartEvent” and “AssignStartEvent” into “Success with no matching condition”:

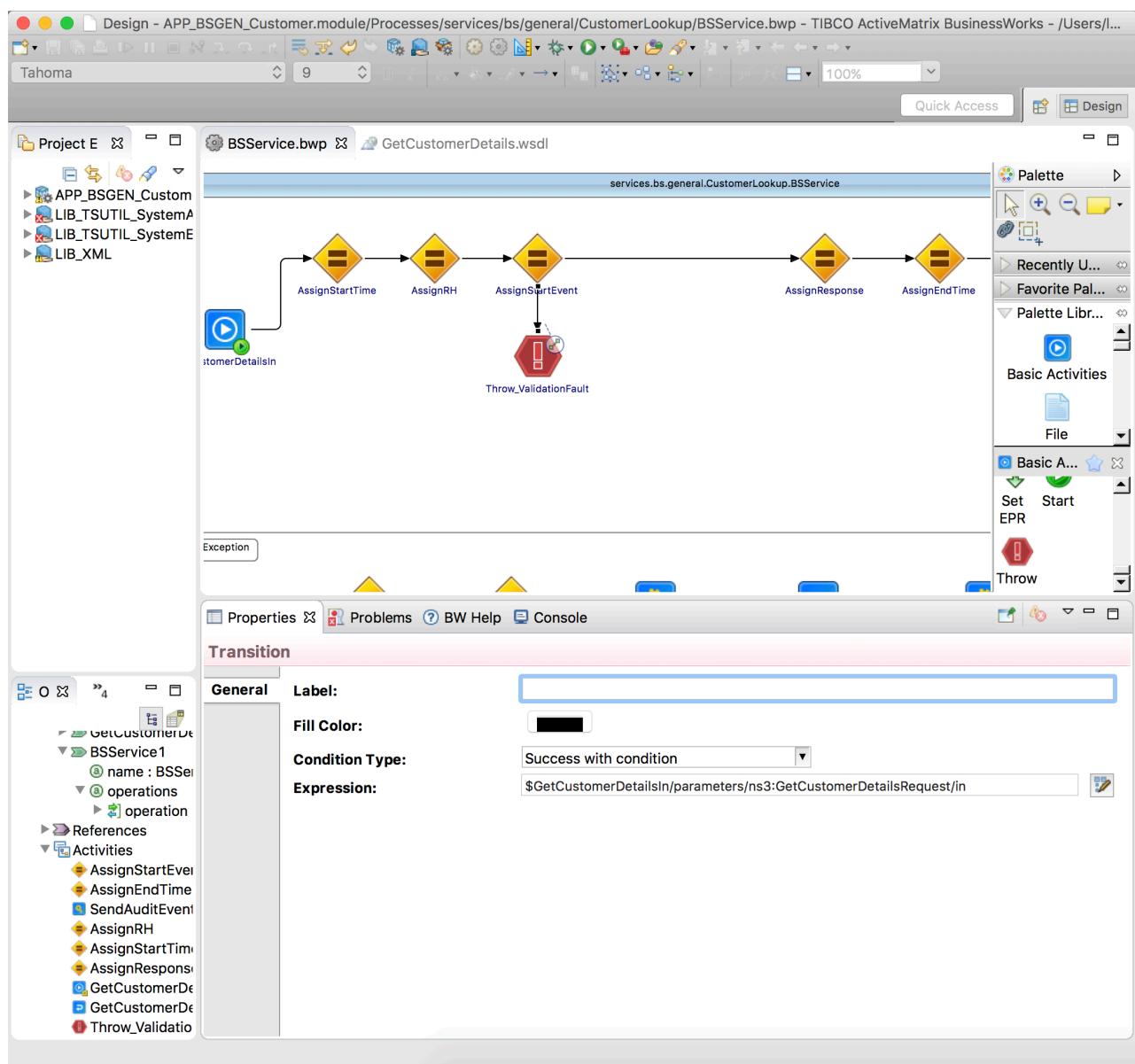


Figure 1: Throw\_ValidationFault sample

7. Change the task “Throw\_ValidationFault” as following:

1. In “Input Editor”, add the XML element “ThrowableValidationFault” from the “LIB\_XML / Schemas / CommonServices / Data / Internal / 1.0 / Throwable” XSD.
2. Change the input mapping as following:
3. ExceptionTimestamp: current-dateTime()
4. ExceptionDescription: a description of the issue, for example concat(“Customer is incorrect: “, \$xxx). When you mention an input element value is incorrect, always put its value in the error description (e.g. \$xxx).

For multiple rules:

- The steps are the same, but you insert a mapping task that will make multiple verifications and output a list of error descriptions. The “Throw\_ValidationFault” task will concatenate all of them into a single ExceptionDescription.

#### 4.4 Finalization

If the service does not invoke any other service via JMS, you must:

1. Delete the resource JNDIClient-ESB01.
2. Delete the resource JNDIClient-ESB01.
3. Delete the property group **resources / services.bs.<area>.<servicename> / Client-ESB01**.

**Notes:**

- Bind the JMS connection to the **north** EMS server if not calling to a technical service.
- Bind the JMS connection to the **south** EMS server if calling to a technical service.
- TSUTIL\_SystemAudit\_Client and TSUTIL\_SystemErrorHandler\_Client use their own JMS connection resources so deleting the resources from the Business service is not an issue. They are the only services to bind to **logging** EMS server.

## 5 Create a Business Service Provider as a Shared Module

This chapter explains how to create a new Business service as a Shared Module, only accessible as process-call from another BW Business service.

### 5.1 Template Overview

The template module APP\_BSServiceTemplate.module allows you to create a new application for a Business service. It contains the following objects:

- Processes:
  - **BSService**: the service implementation.
- Resources:
  - **JNDIClient-ESB01**: the JNDI connection to the ESB Server that the Business service MUST use when invoking other ESB services exposed on SOAP over JMS.

**Notes:**

- 2 The JNDI and JMS connections used by the System Error Handler and System Audit are separated from JNDIClient-ESB01 and JMSClient-ESB01. They are contained in the LIB\_TSUTIL\_SystemErrorHandler\_Client and LIB\_TSUTIL\_SystemAudit\_Client modules.

### 5.2 Pre-Requisites

The XSD and WSDL have been created.

### 5.3 Procedure

#### 5.3.1 Create the Application Module

With Windows Explorer:

1. Copy the Service Application Module template folder “LIB\_BSServiceTemplate.module” into the folder <GIT> / trunk / BW / BusinessServices.
2. Rename the copied folder into “LIB\_BS<Area>\_<ServiceName>”.
3. With a text editor, open the .project file into the copied folder and change the project name at the top from LIB\_BSServiceTemplate.module into LIB\_BS<Area>\_<ServiceName>.

**Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>LIB_BSGEN_ServiceA</name>
    <comment></comment>
...

```

With TIBCO BusinessStudio

4. Start TIBCO BusinessStudio
5. In your workspace, import the following modules:
  - e. LIB\_XML (from <GIT> / trunk / XML)
  - f. LIB\_TSUTIL\_SystemAudit\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - g. LIB\_TSUTIL\_SystemErrorHandler\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - h. ~LIB\_BS<Area>\_<ServiceName> (from <GIT> / trunk / BW / BusinessServices)

All subsequent changes are done on the APP\_<TYPE><Area>\_<ServiceName> module:

6. Go to the module overview.
7. Change the name into “LIB\_BS<Area>\_<ServiceName> Module”:

### 5.3.2 Module Properties, Processes, Service Invocation

The rest of the procedure is the same as for a business service provider in an Application, please refer to the previous chapter for all required procedure information, except for the next sections, which are specific to an shared module.

## 5.4 Finalization

If the service does not invoke any other service via JMS, you must:

4. Delete the resource JNDIClient-ESB01.
5. Delete the resource JNDIClient-ESB01.
6. Delete the property group **resources / services.bs.<area>.<servicename> / Client-ESB01**.

#### Notes:

- Bind the JMS connection to the **north** EMS server if **not** calling to a technical service.
- Bind the JMS connection to the **south** EMS server if calling to a technical service.
- TSUTIL\_SystemAudit\_Client and TSUTIL\_SystemErrorHandler\_Client use their own JMS connection resources so deleting the resources from the Business service is not an issue. They are the only services to bind to **logging** EMS server.

## 6 Create a Technical Service Provider as a Shared Module

This chapter explains how to create a Technical Service as a Shared Module, not exposed on JMS nor HTTP, only accessible as process-call from another BW service.

### 6.1 Template Overview

The template module “APP\_ServiceTemplate.module” (renamed as APP\_TSServiceTemplate.module) allows you to create a new application for a Business service. It contains the following objects:

- Processes:
  - **TSService**: the service implementation.
- Resources:
  - **Client-BACKEND-http**: the HTTP client the Technical service MUST use if the back-end system is exposed on HTTP/HTTPS transport.
  - **Client-BACKEND-jdbc**: the JDBC client the Technical service MUST use if the back-end system is exposed as a database.

**Notes:**

- 3 The JNDI and JMS connections used by the System Error Handler and System Audit are contained in the LIB\_TSUTIL\_SystemErrorHandler\_Client and LIB\_TSUTIL\_SystemAudit\_Client modules.

### 6.2 Pre-Requisites

The XSD and WSDL have been created. **If you are using an existing concrete WSDL, see section 6.5.**

### 6.3 Procedure

#### 6.3.1 Create the Application Module

With Windows Explorer:

1. Copy the Service Application Module template folder “LIB\_TSServiceTemplate.module” into the folder **<GIT> / trunk / BW / TechnicalServices**.
2. Rename the copied folder into “LIB\_TS<Area>\_<ServiceName>”.
3. With a text editor, open the .project file into the copied folder and change the project name at the top from LIB\_TSServiceTemplate.module into LIB\_TS<Area>\_<ServiceName>.

**Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>LIB_TSCMD_GetVehicleDetails</name>
    <comment></comment>
```

...

With TIBCO BusinessStudio

1. Start TIBCO BusinessStudio
2. In your workspace, import the following modules:
  - a. LIB\_XML (from <GIT> / trunk / XML)
  - b. LIB\_TSUTIL\_SystemAudit\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - c. LIB\_TSUTIL\_SystemErrorHandler\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - d. LIB\_TS<Area>\_<ServiceName> (from <GIT> / trunk / BW / TechnicalServices)

All subsequent changes are done on the LIB\_<TYPE><Area>\_<ServiceName> module:

3. Go to the module overview.
4. Change the name into “LIB\_<TYPE><Area>\_<ServiceName> Module”:

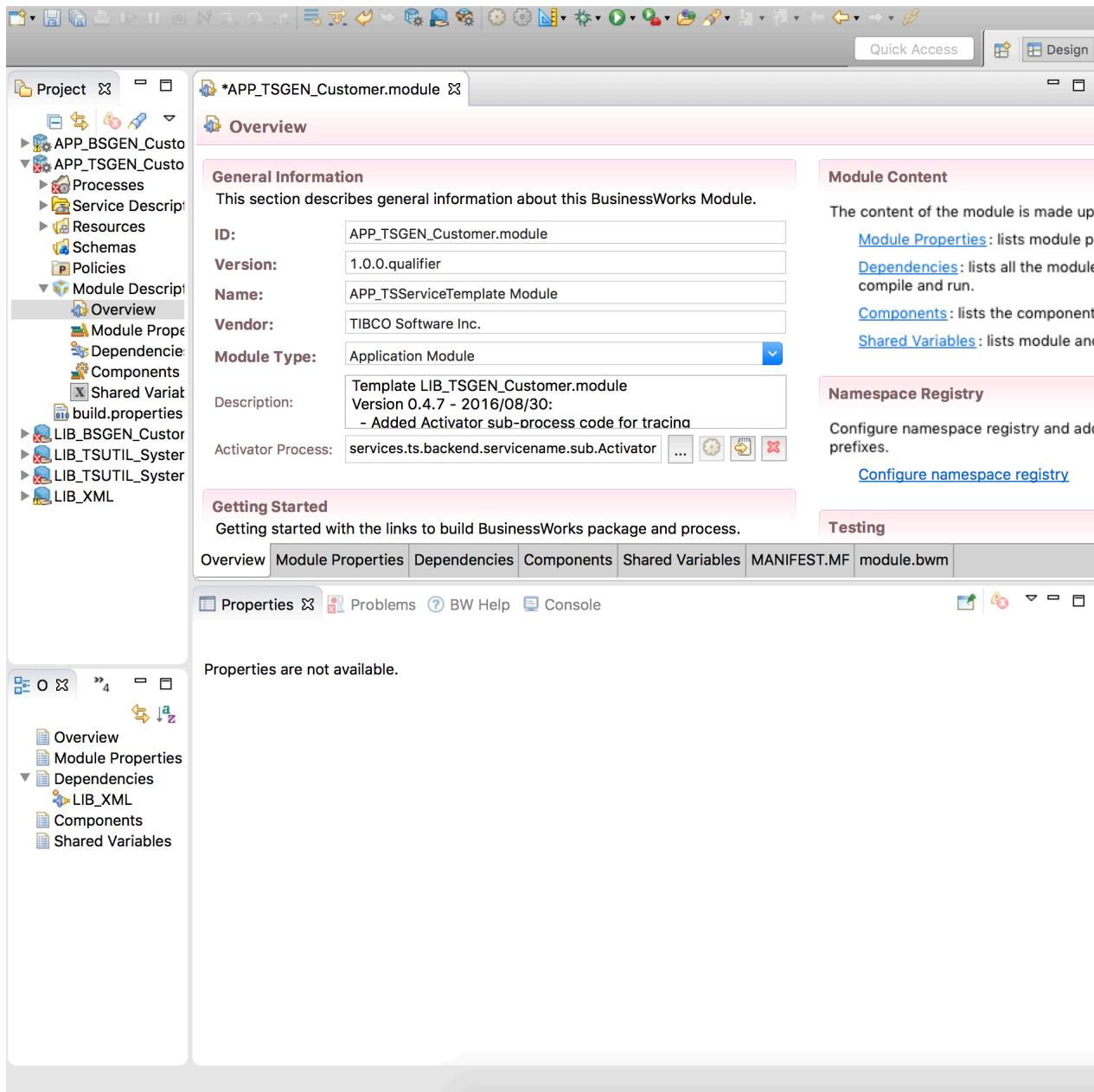


Figure 2: TS Service Provider Shared Module: Module Name

### 6.3.2 Change the Module Properties

1. Go to the module properties.
2. Select the group “services.ts.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.ts.<area>.<servicename>”:

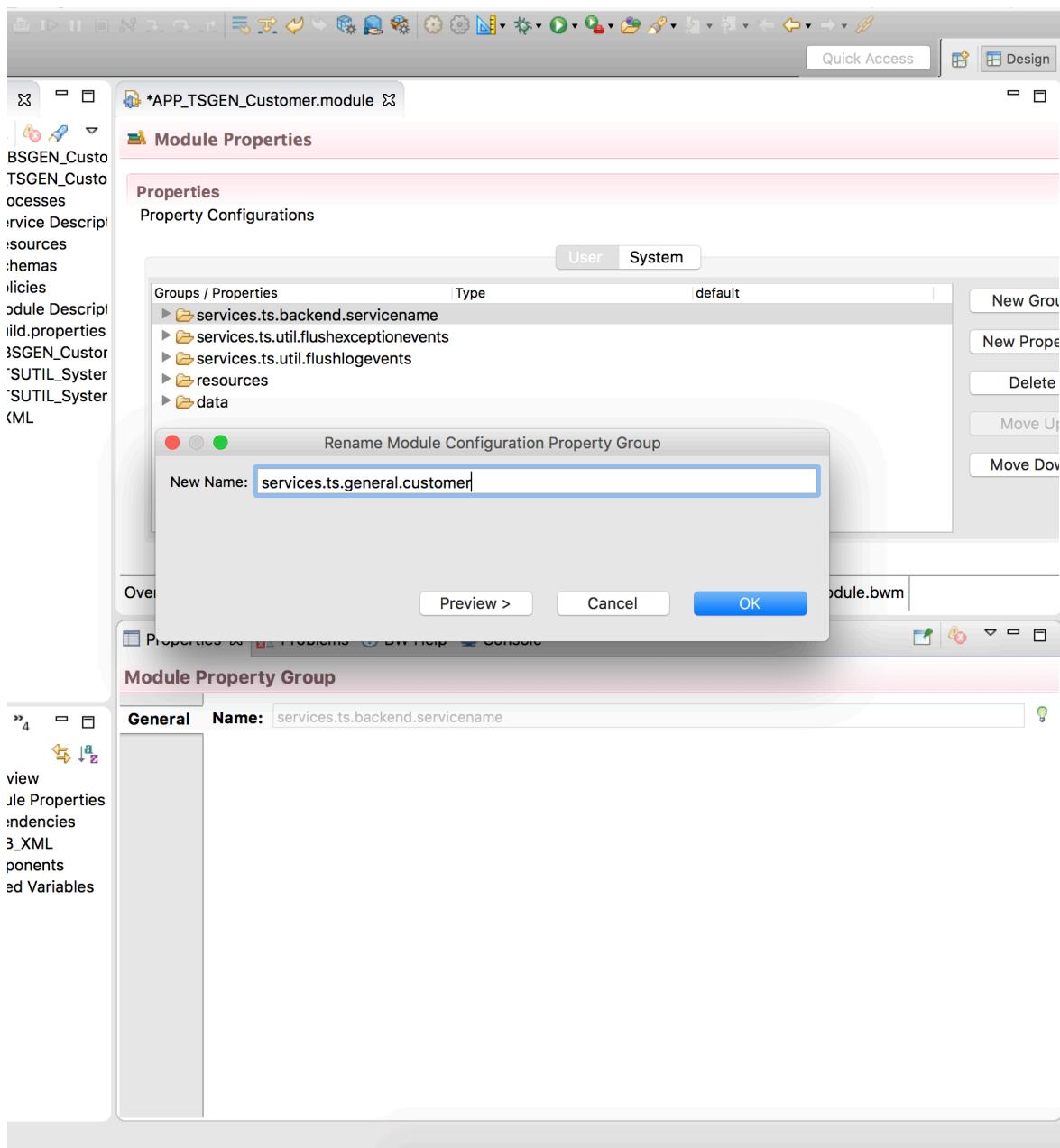
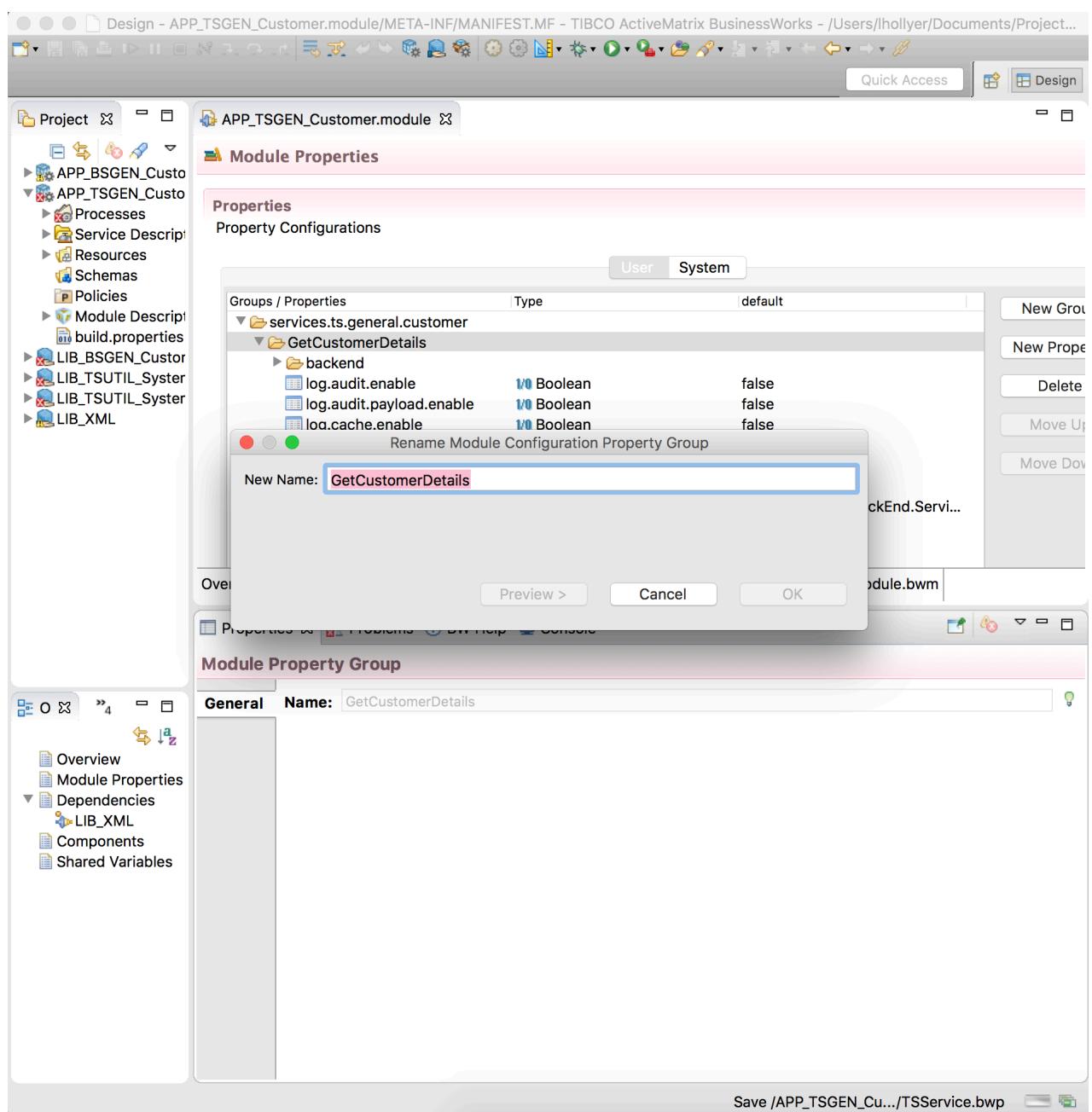


Figure 3: TS Service Provider Shared Module: Module Properties

3. In this group, use the same technique to rename the group “TSService” into “<ServiceName>” (for example “GetCustomerDetails”).



4. Select the group "resources / services.ts.backend.servicename". In the properties pane, use the light-bulb icon to rename the group into "services.ts.<area>.<servicename>":

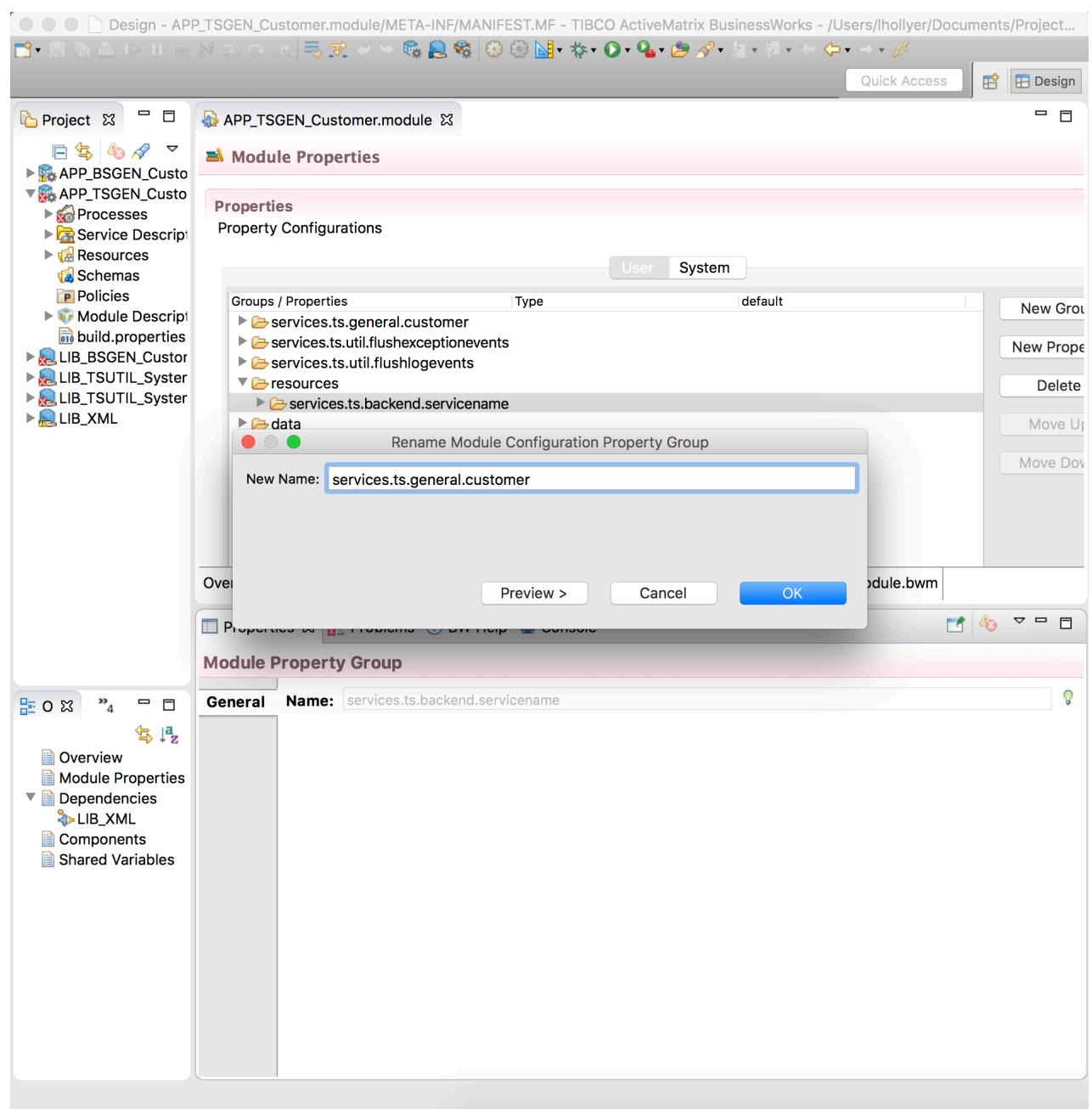


Figure 4: TS Service Provider Shared Module: Resource Properties

### 6.3.3 Create the Process Packages

1. Create the process package “services.ts.<area>.<servicename>”.
2. Move the TSService process into it.
3. Create the process package “services.<type>.<area>.<servicename>.sub”.
4. Move the Activator process into it.

5. Delete the remaining ...servicename... process packages.

#### 6.3.4 Change the Service Process

1. Open the process “services.ts.<area>.<servicename> / TSService”.
2. In the properties, use the light-bulb icon to rename it into “<ServiceName>”:

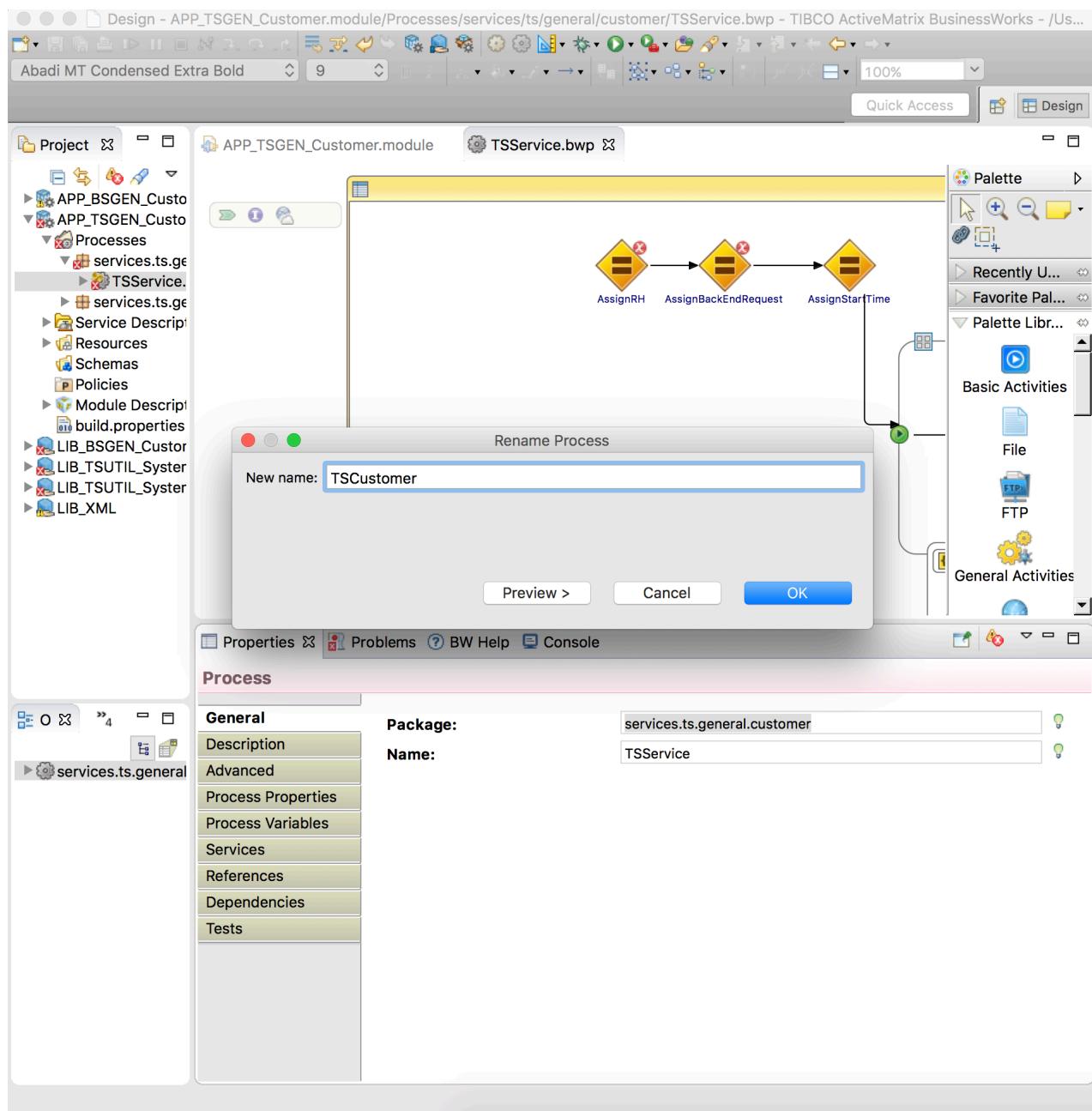


Figure 5: TS Service Provider Shared Module: Process Name

3. Import the XSD and service WSDL of the TS Service into the module.

4. Drag and drop the abstract WSDL into the process:

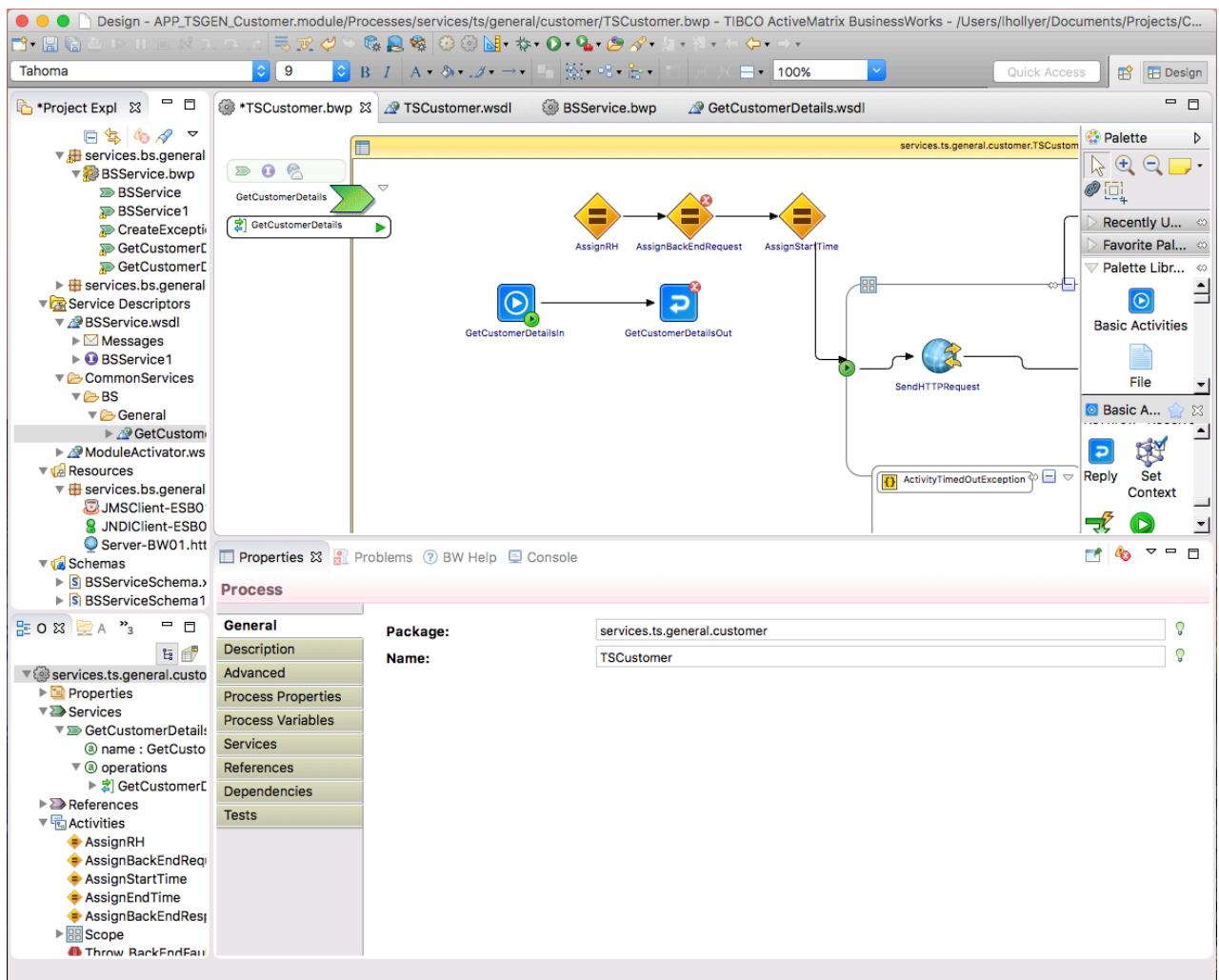


Figure 6: TS Service Provider Shared Module: Implement Service Operation

5. Create a transition between the <Operation>In task and “AssignRH” task:

**N.B.** if there is a conversation error on <Operation>In, click on the <Operation>In task and ensure ‘Create instance’ is checked on the general tab.

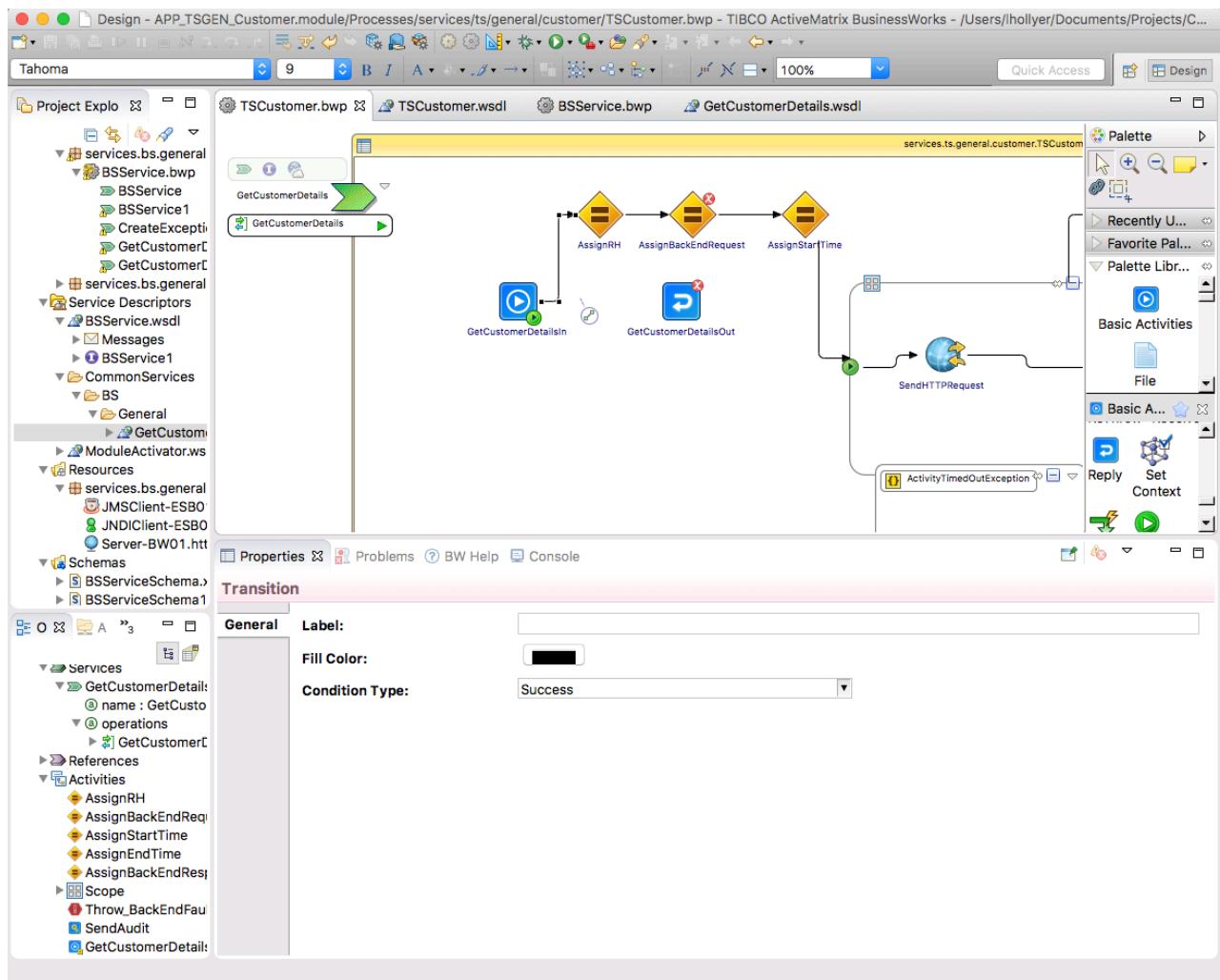


Figure 7: TS Service Provider Shared Module: Change input transition

6. Re-arrange transitions so the <Operation>Out task is located between “AssignBackEndResponse” and “SendAuditEvent” tasks:

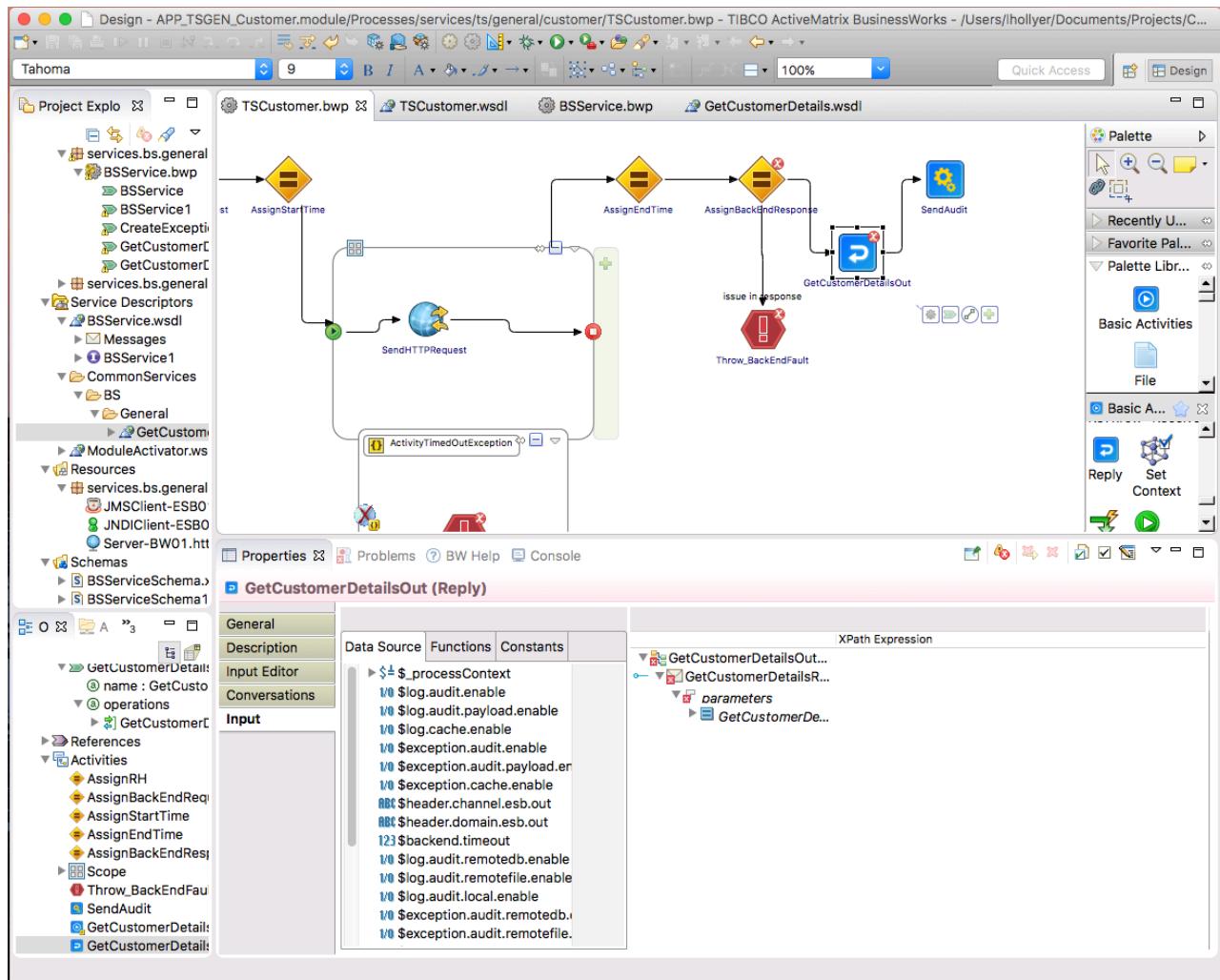


Figure 8: TS Service Provider Shared Module: Reply Transition

7. Save your changes.

### 6.3.5 Fix the Process Mappings

#### 6.3.5.1 AssignRH

This task prepares the message header that can be passed into calls to other ESB services or returned in the reply or fault messages.

Fix the mapping in the task “AssignRH” as following:

1. Add a child variable called “varRequest” and change the formula for the “varRequest” variable into

```
$<Operation>In/parameters/tns9:<Operation>Request
```

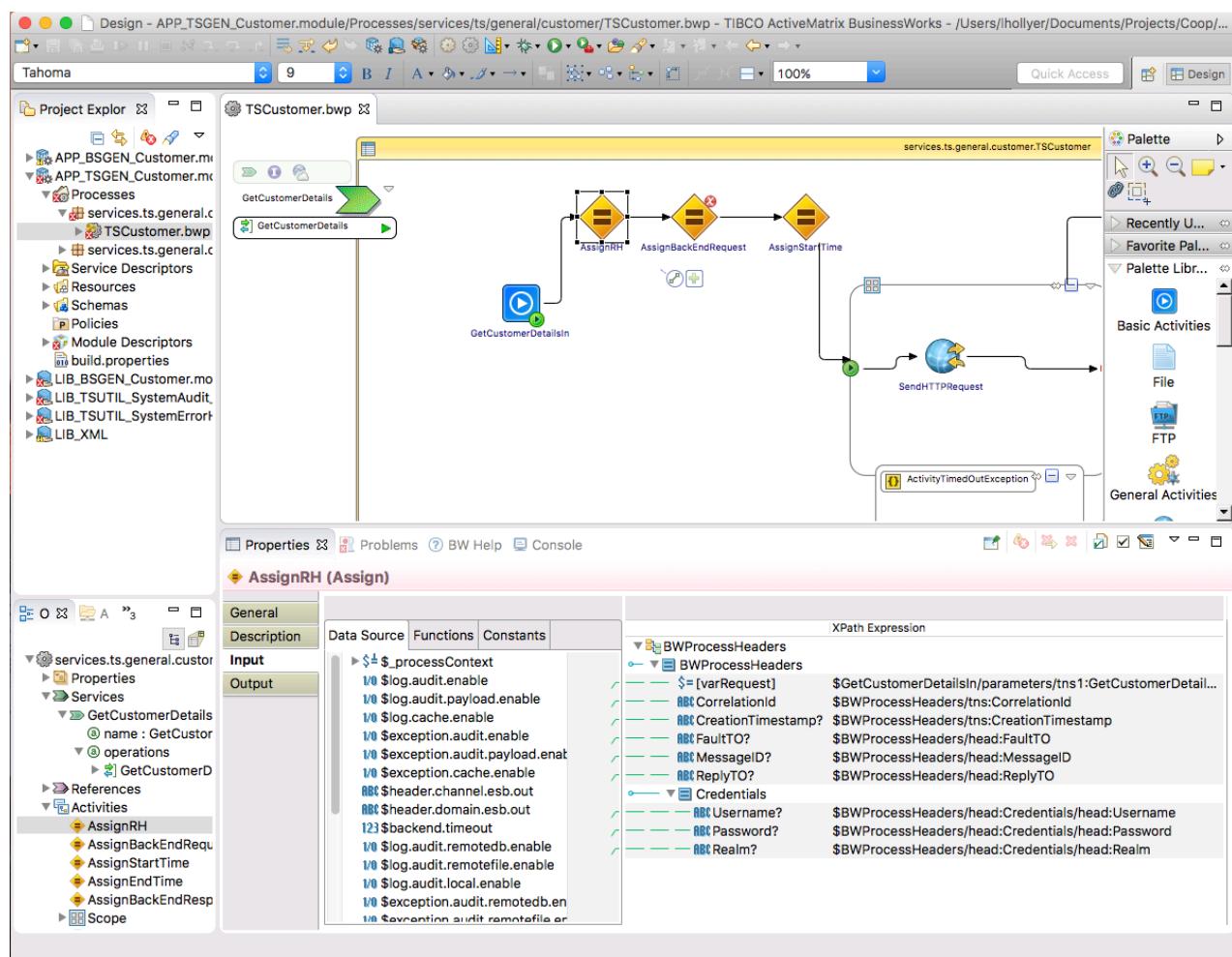


Figure 9: TS Service Provider Shared Module: AssignRH mapping

### 6.3.5.2 Reply\_Fault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML Fault element.

Fix the configuration of the Reply\_Fault task:

1. Select the service, operation and “Fault”.

Fix the mapping of the Reply\_Fault task so:

1. Fault element is a copy of the CreateExceptionEvent / Response / Fault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with no matching condition”.

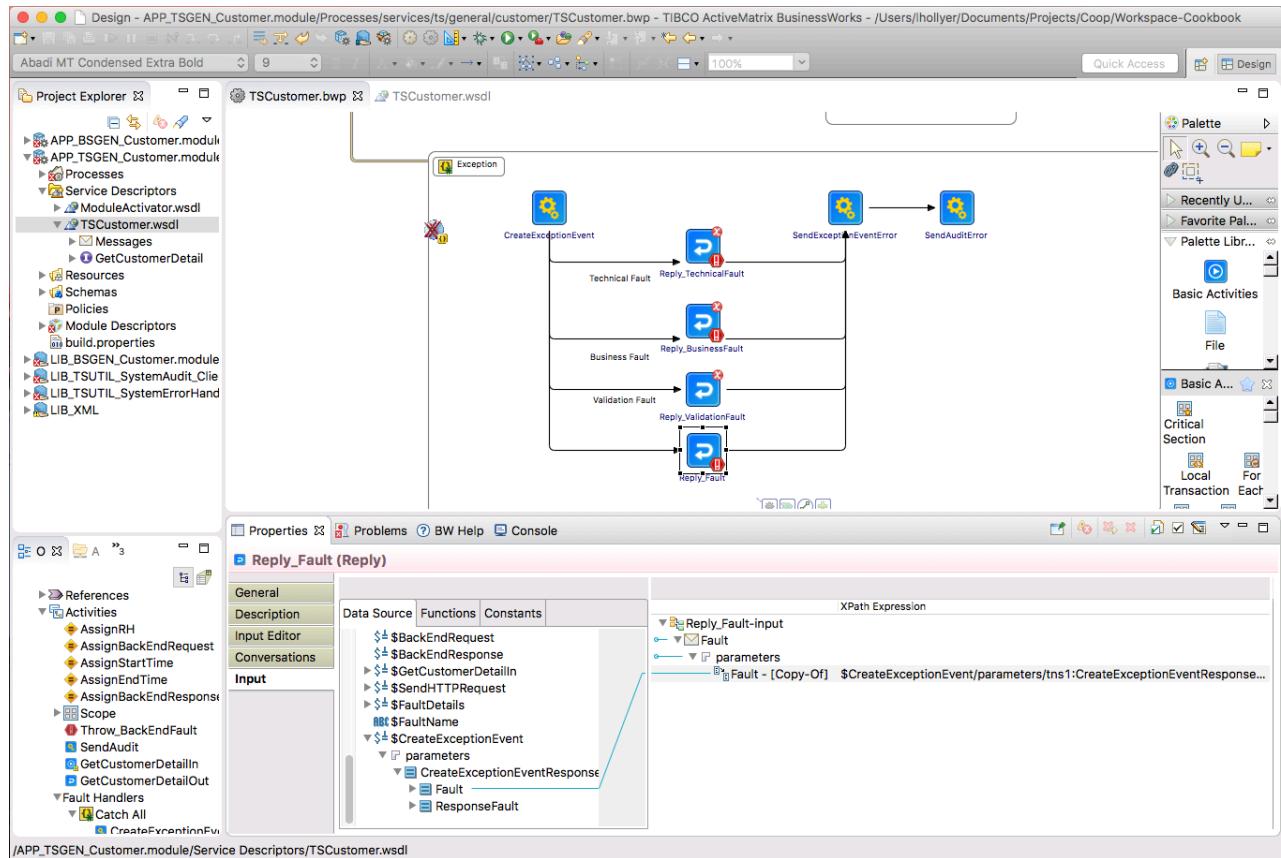


Figure 10: TS Service Provider Shared Module: Reply\_Fault mapping

### 6.3.5.3 Reply\_BusinessFault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML BusinessFault element.

Fix the configuration of the BusinessFault task:

1. Select the service, operation and “BusinessFault”.

Fix the mapping of the BusinessFault task so:

1. BusinessFault element is a copy of the CreateExceptionEvent / Response / BusinessFault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” -

`exists($CreateExceptionEvent/parameters/tns3:CreateExceptionEventResponse/tns3:ResponseFault/tns2:BusinessFault).`

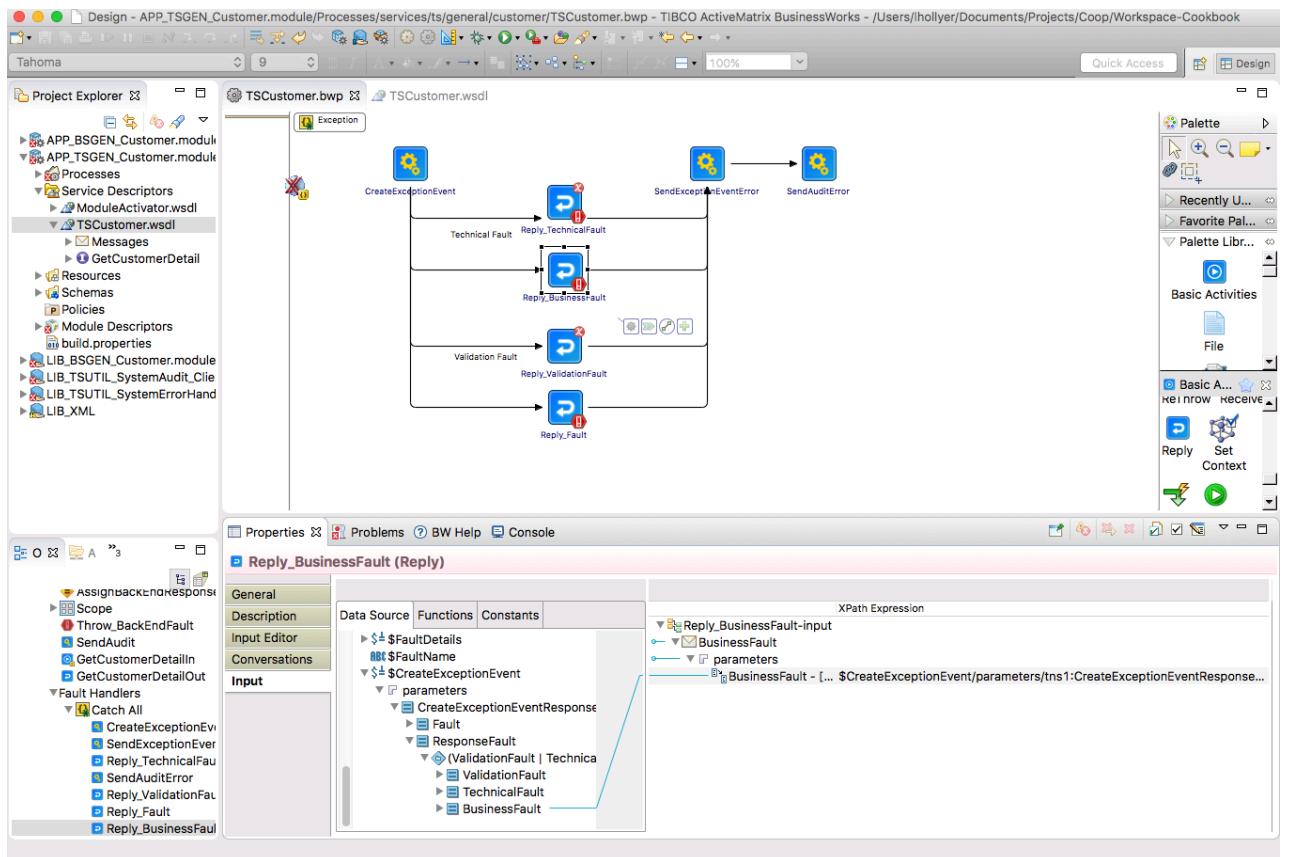


Figure 11: TS Service Provider Shared Module: *Reply\_BusinessFault* mapping

#### 6.3.5.4 Reply\_TechnicalFault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML TechnicalFault element.

Fix the configuration of the TechnicalFault task:

1. Select the service, operation and “TechnicalFault”.

Fix the mapping of the TechnicalFault task so:

1. TechnicalFault element is a copy of the CreateExceptionEvent / Response / TechnicalFault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” –

exists(\$CreateExceptionEvent/parameters/tns3:CreateExceptionEventResponse/tns3:ResponseFault/tns2:TechnicalFault).

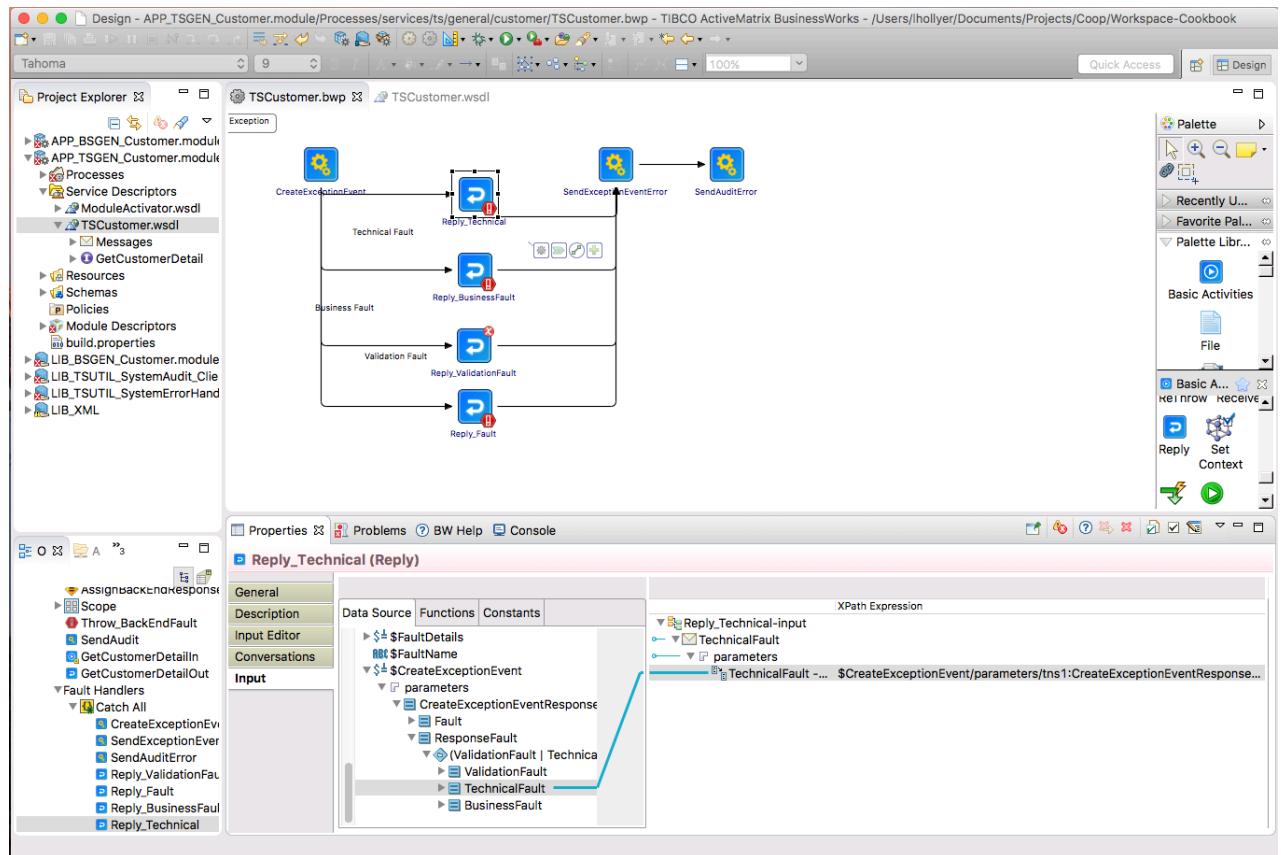


Figure 12: TS Service Provider Shared Module: Reply\_TechnicalFault mapping

### 6.3.5.5 Reply\_ValidationFault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML ValidationFault element.

Fix the configuration of the ValidationFault task:

1. Select the service, operation and “ValidationFault”.

Fix the mapping of the ValidationFault task so:

1. ValidationFault element is a copy of the CreateExceptionEvent / Response / ValidationFault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” –

exists(\$CreateExceptionEvent/parameters/tns3:CreateExceptionEventResponse/tns3:ResponseFault/tns2:ValidationFault).

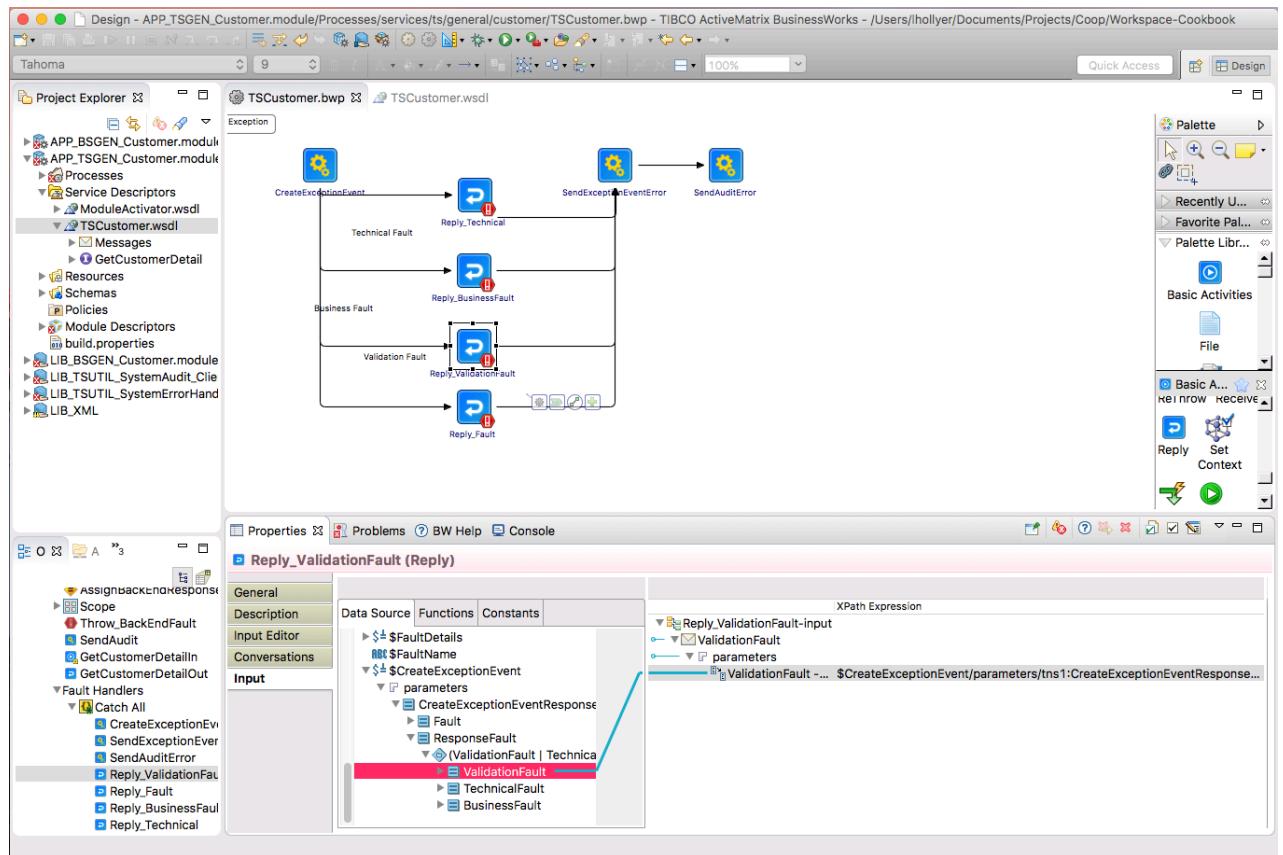
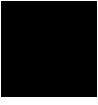


Figure 13: TS Service Provider Shared Module: Reply\_ValidationFault mapping



## CO-OP Bank - ESB – How To: Create BW Services from Templates

#### **6.3.5.6 *SendExceptionEvent***

This ask invokes the ESB Audit service to send one event indicating the request that was sent to the back-end and another one indicating the back-end returned a reply.

Fix the mapping of the ValidationFault task as following:

1. Add a “varBackEnd” child variable and change the value of the “varBackEnd” variable so it contains the name of the back-end system, for example “Experian”. If the back-end is .net, please use the .net service name. If the back-end is a DB you can add the name of the stored procedure as well.

#### **6.3.5.7 *SendAuditEvent\_Error***

This ask invokes the ESB Audit service to send one event indicating the request that was sent to the back-end, another one indicating the back-end returned a reply (if it did) and another one indicating an error happened.

Fix the mapping of the ValidationFault task as following:

1. Add a “varBackEnd” child variable and change the value of the “varBackEnd” variable so it contains the name of the back-end system, for example “Experian”. If the back-end is .net, please use the .net service name. If the back-end is a DB you can add the name of the stored procedure as well.

### 6.3.6 Back End on HTTP/HTTPS

If the back-end system is accessed on HTTP/HTTPS transport, you can modify the service as following.

#### 6.3.6.1 Change the resources

1. Rename the HTTP Client resource from “services.ts.backend.servicename.Client-BACKEND-http” into “services.ts.<area>.<servicename>.Client-<BackEndName>”.
2. Delete the JDBC resource “Client-BACKEND-jdbc”.
3. Delete the old resource package services.ts.backend.servicename.

#### 6.3.6.2 Change the Module Properties

1. Go to the module properties.
2. Select the group “services.ts.<area>.<servicename> / <ServiceName> / backend”. In the properties pane, use the light-bulb icon to rename the group into “<BackEndName>”.

**Example:**

```
services.ts.<area>.<servicename> / <ServiceName> / backend / becomes
services.ts.bline.getcardscheme / GetCardScheme / BLINE
```

3. In the group you renamed, change the “endpoint.uri” value to correspond to the back-end URI.
4. Select the group “resources / services.ts.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.ts.<area>.<servicename>”.
5. Select the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-http”. In the properties pane, use the light-bulb icon to rename the group into “Client-<BackEndName>”.
6. Delete the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-jdbc”.

#### 6.3.6.3 Change Service Process (if back-end is accessed via SOAP)

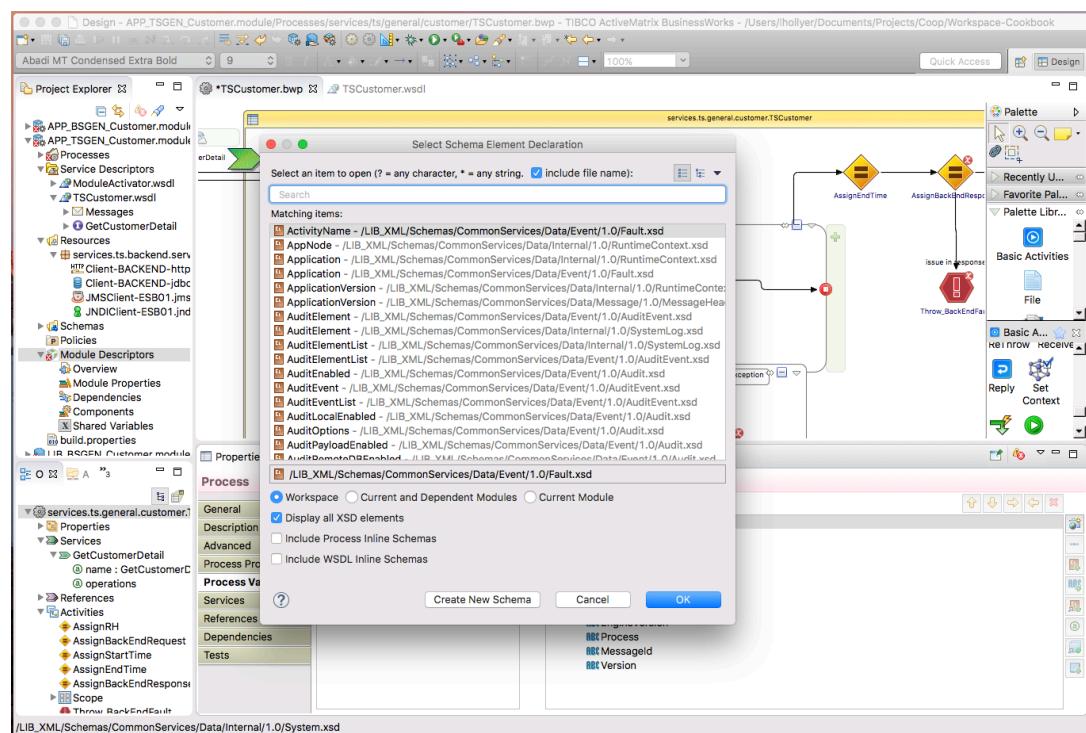
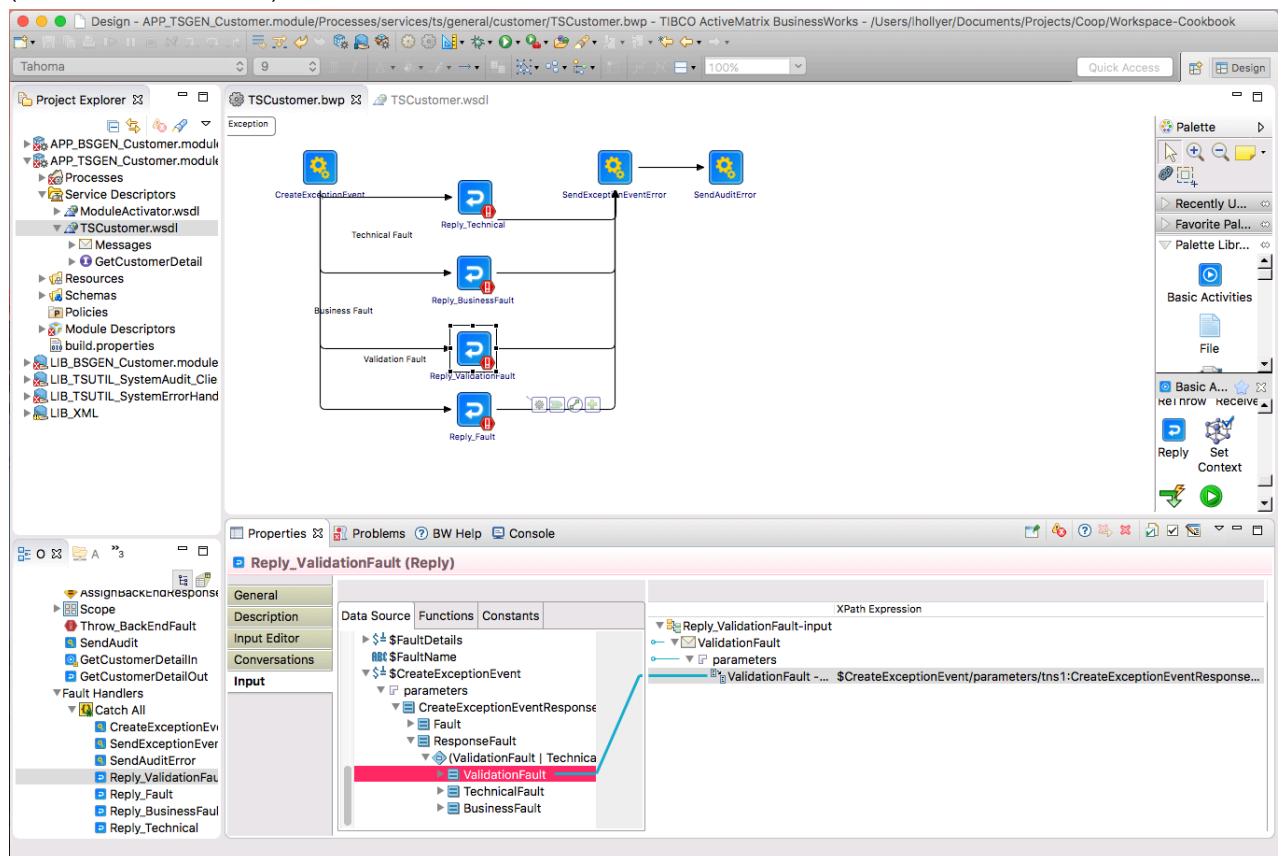
1. Import the concrete WSDL into the “Service Descriptors” folder.

Change the service process as following:

##### Service Invocation

1. Remove the “SendHTTPRequest” task and replace it with an Invoke task.
2. Configure the Service Reference and add a SOAP binding which uses:
  - a. The HTTP client resource “services.ts.<area>.<servicename>.Client-<BackEndName>”.
  - b. The property “services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / endpoint.uri.” for the endpoint URI.

3. Edit the process variable “\$BackEndRequest” so it uses the XML schema of the back-end operation request (from its concrete WSDL).



4. Edit the process variable “\$BackEndResponse” so it uses the XML schema of the back-end operation response (from its concrete WSDL).
5. Map the request to the back-end in the “AssignBackEndRequest”. This task stores the request into the process variable \$BackEndRequest so it can be re-used in other tasks (for the SystemAudit tasks for example).
6. Change the mapping of the Invoke task so it re-uses the \$BackEndRequest variable.
7. Map the response from the back-end in the “AssignBackEndResponse” task. This task stores the response into the process variable \$BackEndResponse so it can be re-used in other tasks (for the SystemAudit tasks for example).

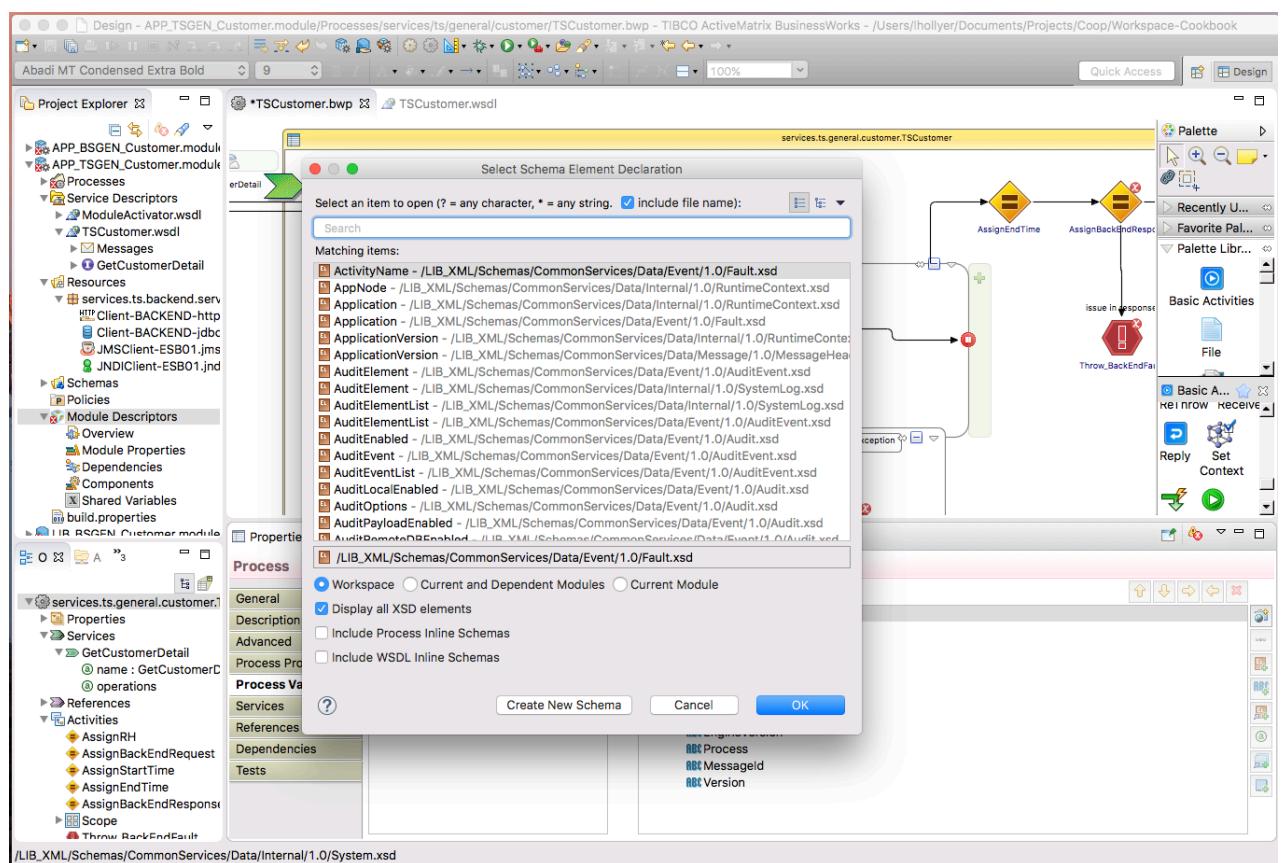
#### **6.3.6.4 Change Service Process (*if back-end is accessed via raw HTTP*)**

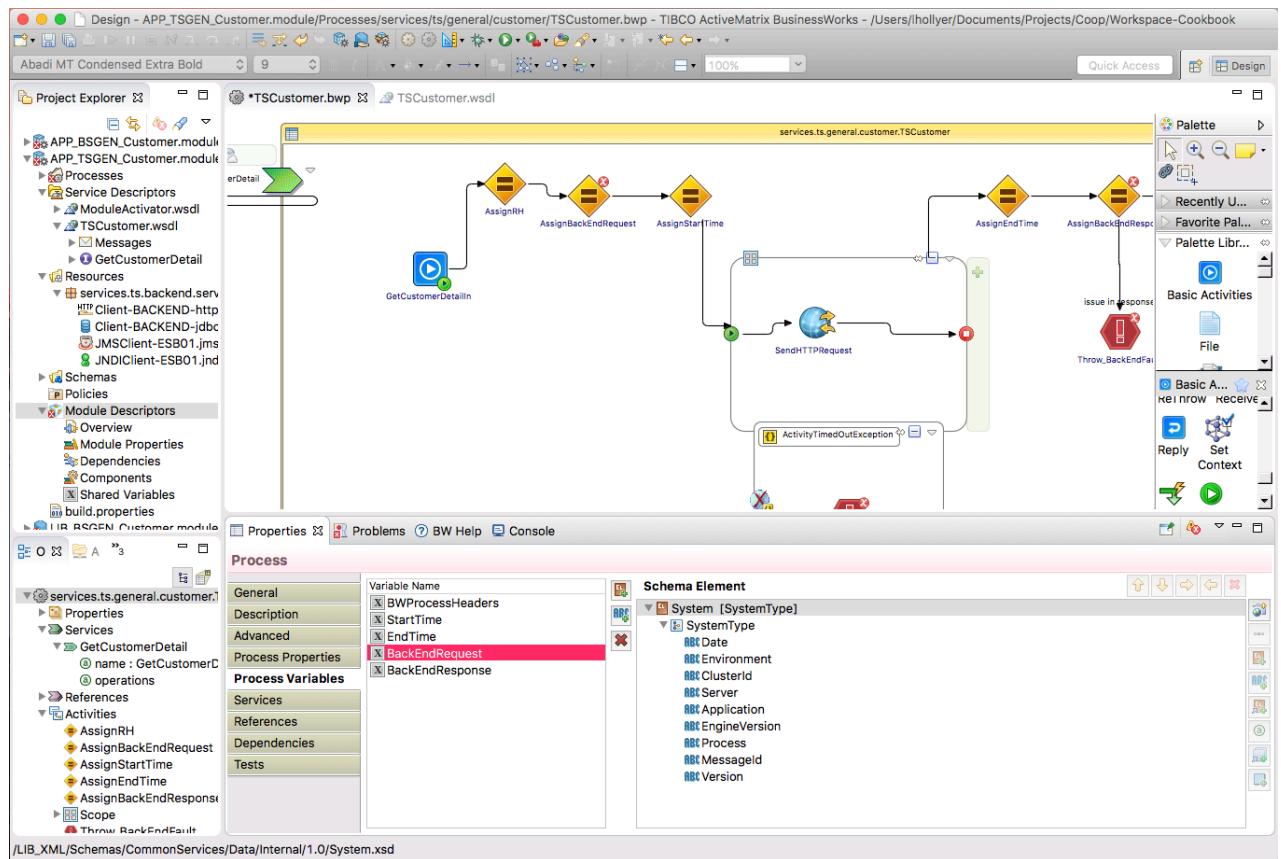
Change the service process as following:

##### **Service Invocation**

1. Edit the process variable “\$BackEndRequest” so it uses the XML schema of the back-end operation request.

## CO-OP Bank - ESB – How To: Create BW Services from Templates





2. Edit the process variable “\$BackEndResponse” so it uses the element “Payload” from the **LIB\_XML / Schemas / CommonServices / Data / Event / 1.0 / AuditEvent.xsd** XSD.
3. Map the request to the back-end in the “AssignBackEndRequest”. This task stores the request into the process variable \$BackEndRequest so it can be re-used in other tasks (for the SystemAudit tasks for example).
4. Change the mapping of the Invoke task so it re-uses the \$BackEndRequest variable.
5. Map the \$SendHTTPRequest variable into the input the “AssignBackEndResponse” task. This task stores the response into the process variable \$BackEndResponse so it can be re-used in other tasks (for the SystemAudit tasks for example).

#### **6.3.6.5 Change Service Process (all cases)**

##### **Detect Back End Errors**

If the service must analyze the back-end reply to detect back-end errors, this can be done in the “Throw\_BusinessFault” task and the transition from “AssignBackEndResponse” task to “Throw\_BusinessFault” task. In the “Throw\_BusinessFault” task you can change the text and description of the issue.

If not, you can delete the “Throw\_BusinessFault” task.

If the service operation can throw various SOAP faults that can be mapped to an ESB BusinessFault, you can handle them as following:

5. Right-click on the Invoke task and in the contextual menu, select the option “Catch / <fault>” where <fault> if the SOAP fault to handle.
6. In the new Catch block, add a “Throw” task.
7. In “Input Editor” of the “Throw” task, add the XML element “ThrowableBusinessFault” from the “Schemas / CommonServices / Data / Message / Internal / Throwable” XSD.
8. Change the mapping of the task to populate the text and description of the fault.

## Timeout Faults

1. In the “Throw\_TimeoutFault” task, you can change the text and description of the timeout error.

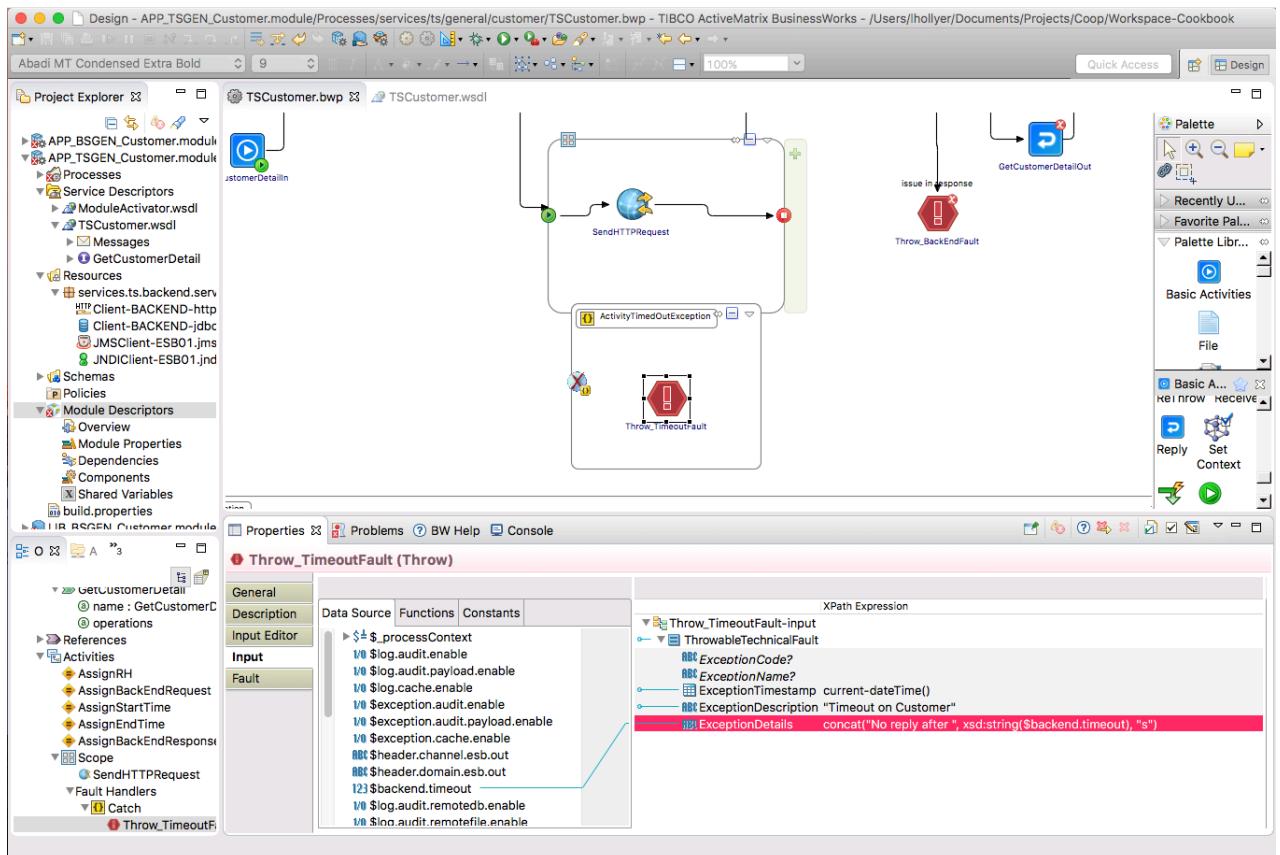


Figure 14: TS Service Provider Shared Module: Throw\_TimeoutFault mapping

2. If the task has no input, fix it as following: in “Input Editor” add the XML element “ThrowableBusinessFault” from the “Schemas / CommonServices / Data / Internal / 1.0 / Internal / Throwable” XSD:

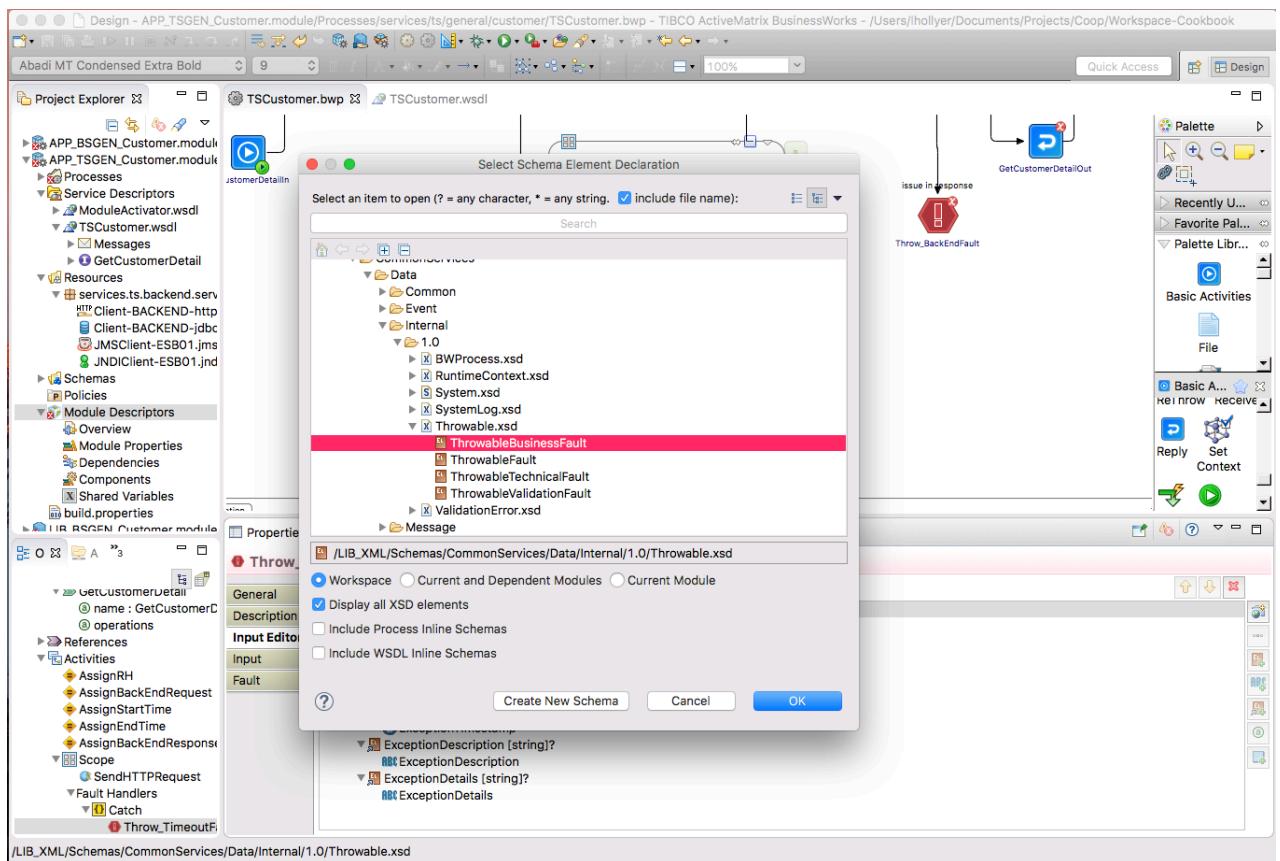


Figure 15: TS Service Provider Shared Module: Throw\_TimeoutFault configuration

3. Delete and recreate the task if necessary.

#### 6.3.6.6 Add HTTPS Resources

If the back-end is accessed via HTTPS, the following resources must be added manually and named as such:

- Resources:
  - SSL Client Configuration: `services.ts.<area>.<servicename>.Client-<BackEndName>-ssl`
  - Keystore Provider Resource: `services.ts.<area>.<servicename>.Client-<BackEndName>-jks`
- Module Properties
  - Keystore URL: `services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / keystore.url.`
  - Keystore Password: `services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / keystore.password.`
  - Refresh Interval: `services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / keystore.refresh.interval` (3600000 ms as default value)

### 6.3.7 Back End on JDBC

If the back-end system is a database, you can modify the service as following.

#### 6.3.7.1 Change the resources

1. Rename the JDBC Client resource from “services.ts.backend.servicename.Client-BACKEND-jdbc” into “services.ts.<area>.<servicename>.Client-<BackEndName>”.
2. Delete the HTTP resource “Client-BACKEND-http”.
3. Delete the old resource package services.ts.backend.servicename.

#### 6.3.7.2 Change the Module Properties

1. Go to the module properties.
2. Select the group “services.ts.<area>.<servicename> / <ServiceName> / backend / BACKEND”. In the properties pane, use the light-bulb icon to rename the group into “<BackEndName>”.
3. In the group you renamed, delete the “endpoint.uri” property.
4. Select the group “resources / services.ts.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.ts.<area>.<servicename>”.
5. Select the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-jdbc”. In the properties pane, use the light-bulb icon to rename the group into “Client-<BackEndName>”.
6. Delete the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-http”.

#### 6.3.7.3 Change Service Process

Change the service process as following:

1. Remove the “SendHTTPRequest” task and replace it with a JDBC task.
2. Configure the JDBC tasks so it uses:
  - a. The JDBC client resource “services.ts.<area>.<servicename>.Client-<BackEndName>”.
  - b. The property “services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / timeout” for the timeout.
3. Map the request to the back-end in the “AssignBackEndRequest”. This task stores the request into the process variable \$BackEndRequest so it can be re-used in other tasks (for the SystemAudit tasks for example).
4. Map the response from the back-end in the “AssignBackEndResponse” task. This task stores the response into the process variable \$BackEndResponse so it can be re-used in other tasks (for the SystemAudit tasks for example).
5. If the service must analyze the back-end reply to detect back-end errors, this can be done in the “Throw\_BusinessFault” task and the transition from “AssignBackEndResponse” task to “Throw\_BusinessFault” task. In the “Throw\_BusinessFault” task you can change the text and description of the issue.

If not, you can delete the “Throw\_BusinessFault” task.

6. In the “Throw\_TimeoutFault” task, you can change the text and description of the timeout error.

## 6.4 Implement the Service

### 6.4.1 Service Response

In the <Operation>Out task, you MUST map the ResponseHeader element as following:

- **CorrelationId:** \$BWProcessHeaders / CorrelationId
- **MessageID:** \$BWProcessHeaders / MessageID
- **Version:** \$BWProcessHeaders / Response / Version
- **ResponseDateTime:** current-dateTime()

**N.B.** if <Operation>Out does not have the ResponseHeader element, check the WSDL location imports the Schema in LIB\_TS<Service>.module's XSD, and check the XSD has the ResponseHeader set to ResponseHeaderType from LIB\_XML.

### 6.4.2 Invoking another ESB service

If you invoke another ESB service (BS or TS), you MUST map the RequestHeader element as following:

- **CorrelationId:** \$BWProcessHeaders / CorrelationId
- **MessageID:** \$BWProcessHeaders / MessageID
- **Version:** The version of the invoked service, for example “1.0”.
- **RequestDateTime:** current-dateTime()

**N.B.** if <Operation>Out does not have the ResponseHeader element, check the WSDL location imports the Schema in LIB\_TS<Service>.module's XSD, and check the XSD has the RequestHeader set to RequestHeaderType from LIB\_XML.

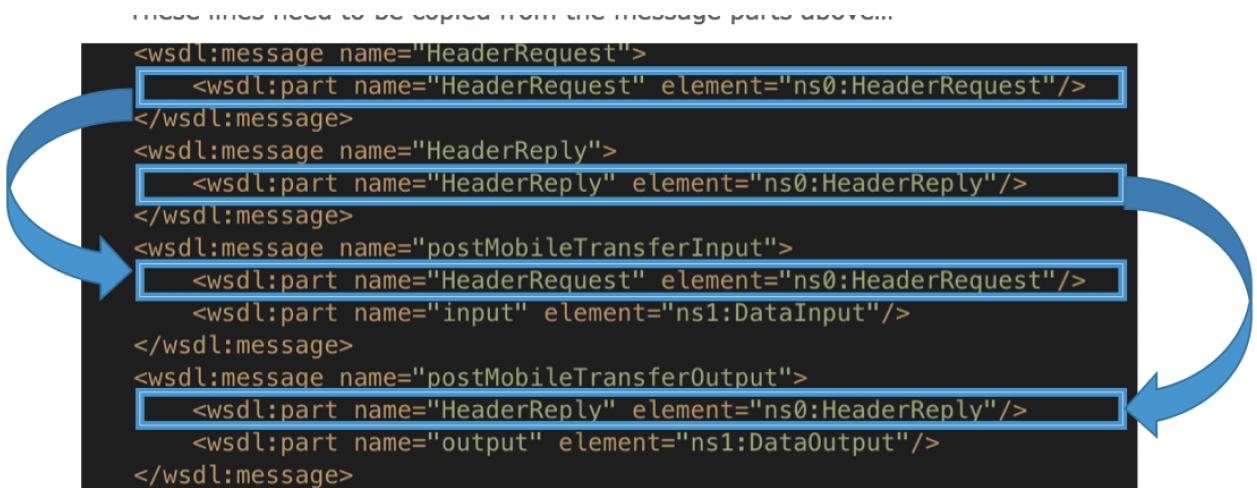
### 6.4.3 Validate the Request

Some service may require additional input data validation (besides what is defined inside the service schema, which BW validates by default). In such case, you must implement the validation as following:

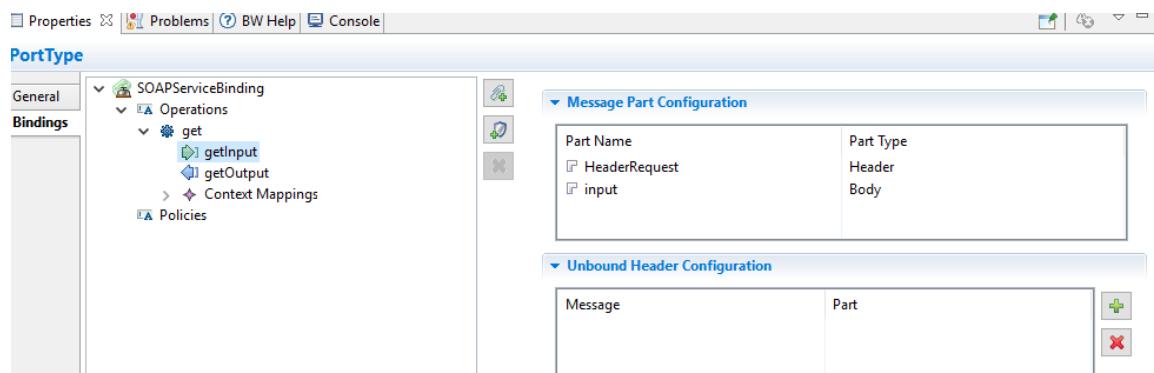
- The steps are the same as for a business service (see section 4.3.5), but the “Throw\_ValidationFault” happens between the “AssignRH” task and “AssignBackEndRequest” task.

## 6.5 Using an Existing Concrete WSDL

1. Duplicate the lines highlighted in the concrete WSDL file:



2. Follow the previous steps as normal to import the WSDL into the process.
3. In Module Descriptors / Components, select the binding and click on “Bindings” in the properties pane.
4. Change the input operation HeaderRequest to be of type “Header” instead of “Body”:



5. Change the output operation HeaderResponse to be of type “Header” too.
6. Continue through the steps as normal.

## 7 Create a Technical Service Provider as an Application

This chapter explains how to create a new Technical Service exposed on a SOAP over JMS binding, on a JMS queue.

### 7.1 Template Overview

The template module “APP\_ServiceTemplate.module” (renamed as APP\_TSServiceTemplate.module) allows you to create a new application for a Business service. It contains the following objects:

- Processes:
  - **TSService**: the service implementation.
  - **Activator**: the process executed when the application starts up inside an appnode. This can be used to trace configuration settings (such as back-end URLs) or call other Activator required in dependent modules.
- WSDLs:
  - **ModuleActivator.wsdl**: the abstract WSDL of the **Activator** process (this one should never be changed at all: it is created by BW when the Activator process is created).
- Resources:
  - **Client-BACKEND-http**: the HTTP client the Technical service MUST use if the back-end system is exposed on HTTP/HTTPS transport.
  - **Client-BACKEND-jdbc**: the JDBC client the Technical service MUST use if the back-end system is exposed as a database.
  - **JNDIClient-ESB01**: the JNDI connection to the ESB Server that the Business service MUST use when invoking other ESB services exposed on SOAP over JMS.

**Notes:**

- 4 The JNDI and JMS connections used by the System Error Handler and System Audit are separated from JNDIClient-ESB01 and JMSClient-ESB01. They are contained in the LIB\_TSUTIL\_SystemErrorHandler\_Client and LIB\_TSUTIL\_SystemAudit\_Client modules.

### 7.2 Pre-Requisites

The XSD and WSDL have been created.

#### 7.2.1 Create the Application Module

With Windows Explorer:

1. Copy the Service Application Module template folder “APP\_TSServiceTemplate.module” into the folder <GIT> / trunk / BW / TechnicalServices.
2. Rename the copied folder into “APP\_TS<Area>\_<ServiceName>”.

3. With a text editor, open the .project file into the copied folder and change the project name at the top from APP\_TSServiceTemplate.module into APP\_TS<Area>\_<ServiceName>.

**Example:**

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>APP_TSCMD_GetVehicleDetails</name>
    <comment></comment>
    ...

```

With TIBCO BusinessStudio

4. Start TIBCO BusinessStudio
5. In your workspace, import the following modules:
  - a. LIB\_XML (from <GIT> / trunk / XML)
  - b. LIB\_TSUTIL\_SystemAudit\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - c. LIB\_TSUTIL\_SystemErrorHandler\_Client (from <GIT> / trunk / BW / TechnicalServices)
  - d. LIB\_TS<Area>\_<ServiceName> (from <GIT> / trunk / BW / TechnicalServices)

All subsequent changes are done on the APP\_<TYPE><Area>\_<ServiceName> module:

5. Go to the module overview.
6. Change the name into “APP\_<TYPE><Area>\_<ServiceName> Module”:

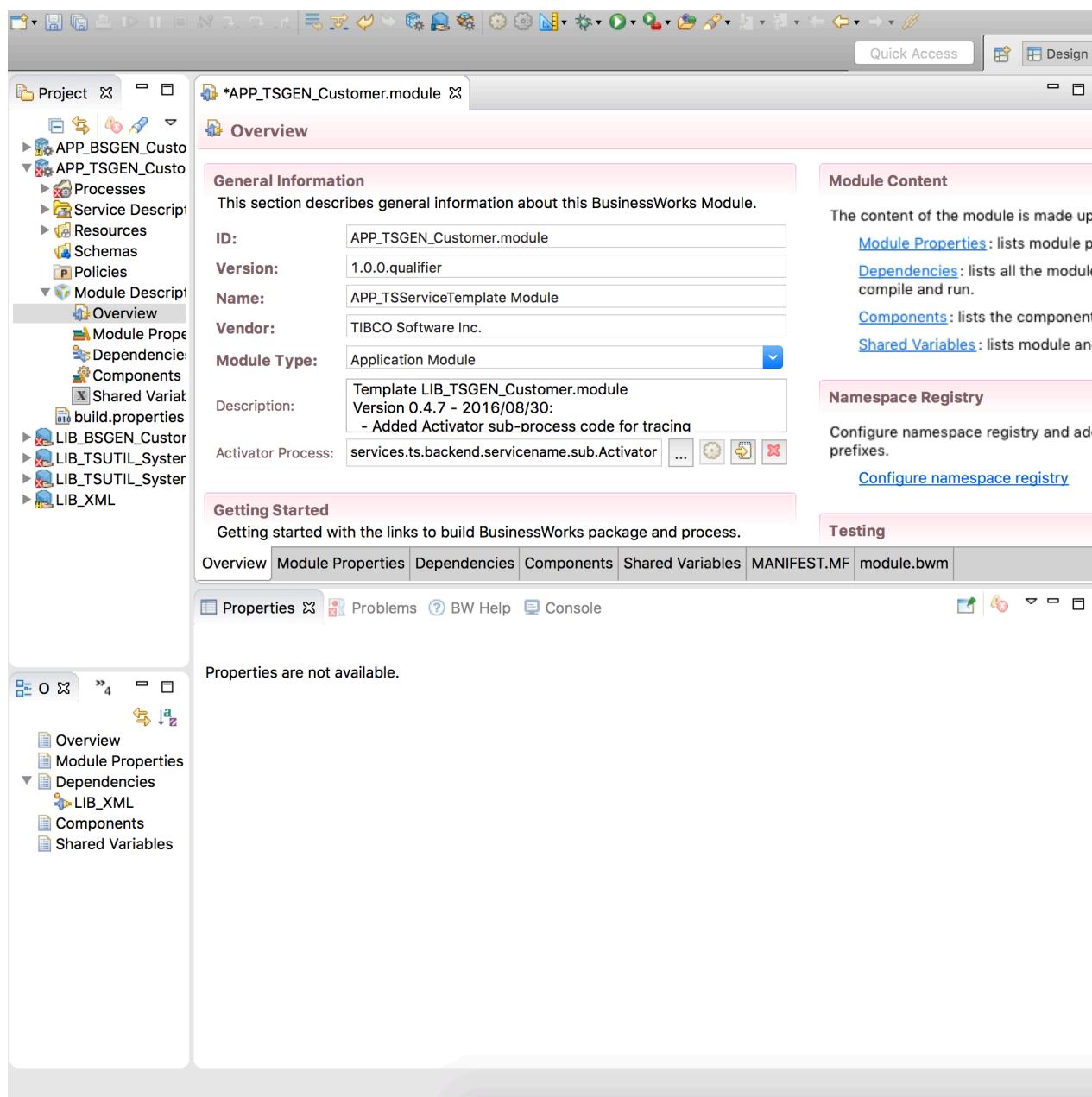


Figure 16: TS Service Provider Shared Module: Module Name

### 7.2.2 Change the Module Properties

5. Go to the module properties.
6. Select the group “services.ts.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.ts.<area>.<servicename>”:

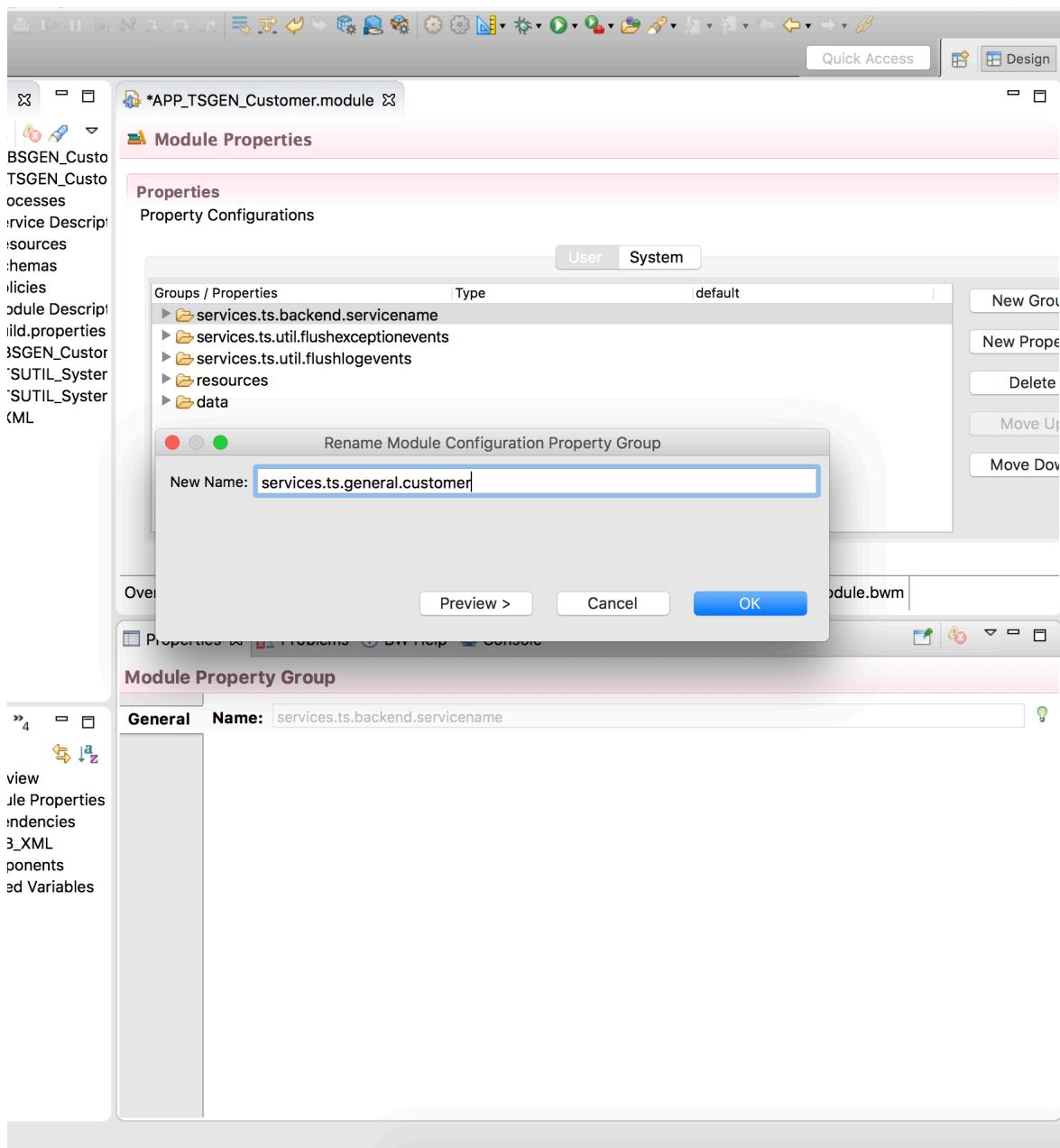
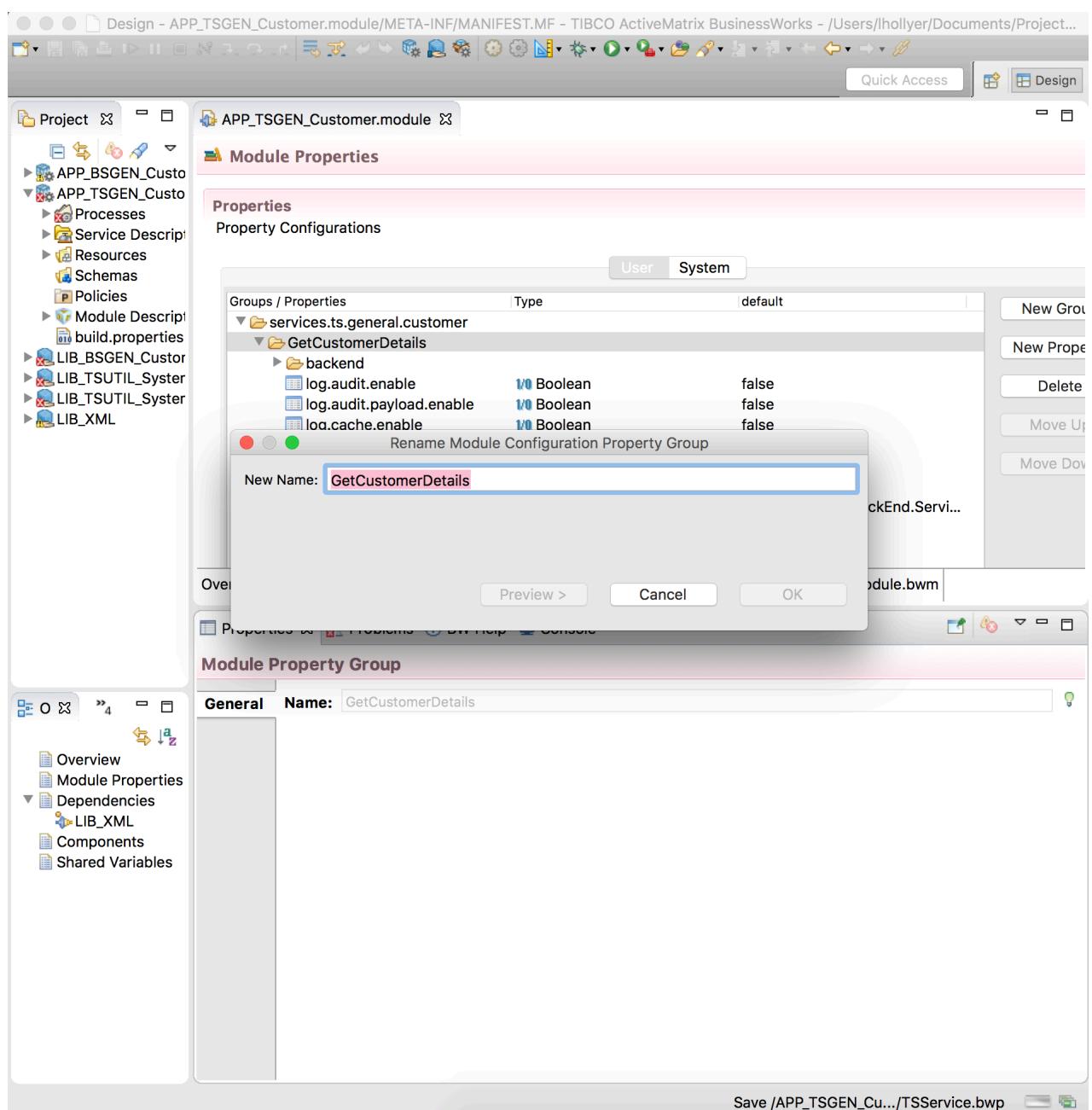


Figure 17: TS Service Provider Shared Module: Module Properties

7. In this group, use the same technique to rename the group “TSService” into “<ServiceName>” (for example “GetCustomerDetails”).



8. Select the group "resources / services.ts.backend.servicename". In the properties pane, use the light-bulb icon to rename the group into "services.ts.<area>.<servicename>":

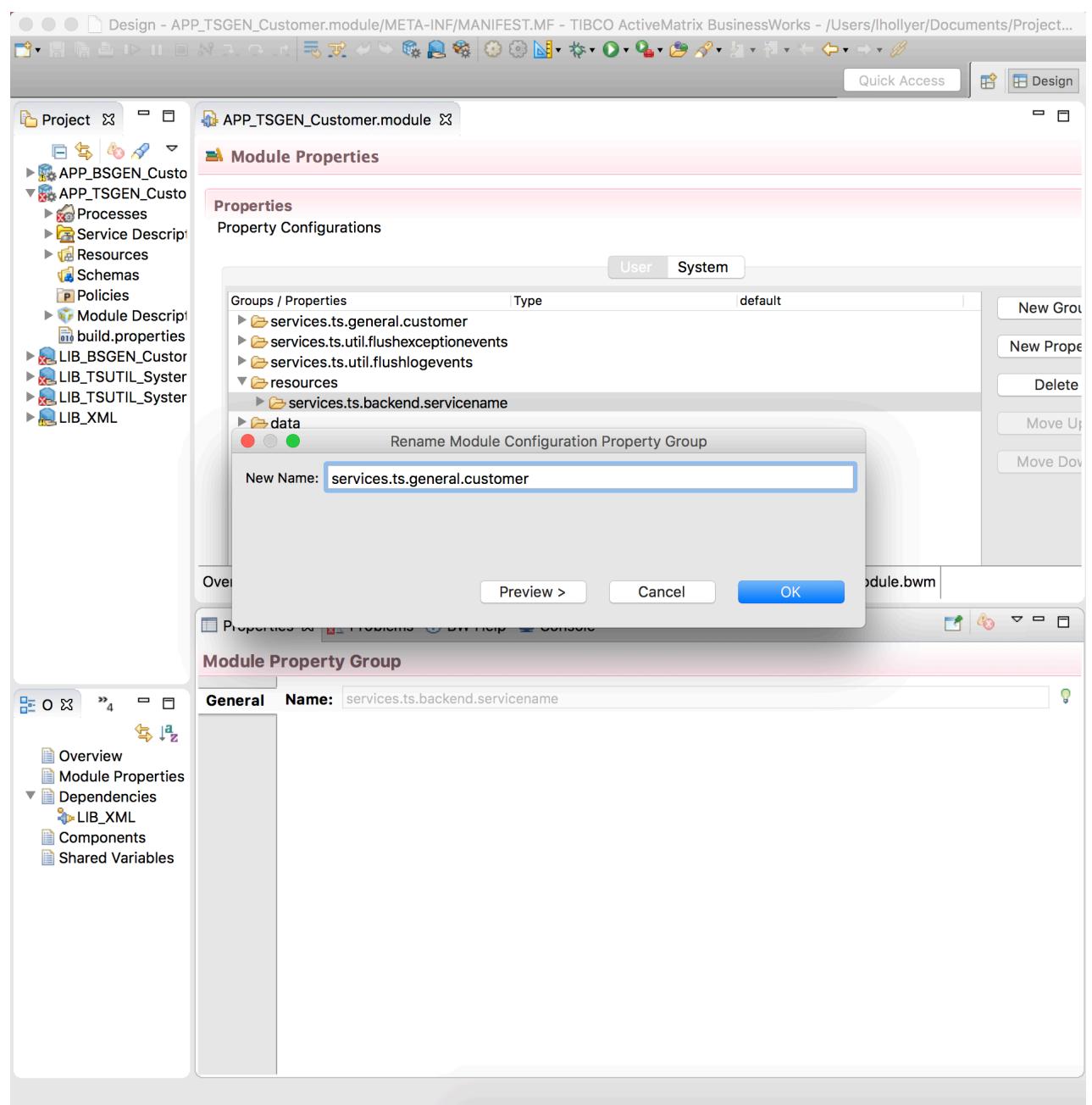


Figure 18: TS Service Provider Shared Module: Resource Properties

### 7.2.3 Create the Process Packages

6. Create the process package “services.ts.<area>.<servicename>”.
7. Move the TSService process into it.
8. Create the process package “services.<type>.<area>.<servicename>.sub”.
9. Move the Activator process into it.

10. Delete the remaining ...servicename... process packages.

#### 7.2.4 Change the Service Process

8. Open the process “services.ts.<area>.<servicename> / TSService”.
9. In the properties, use the light-bulb icon to rename it into “<ServiceName>”:

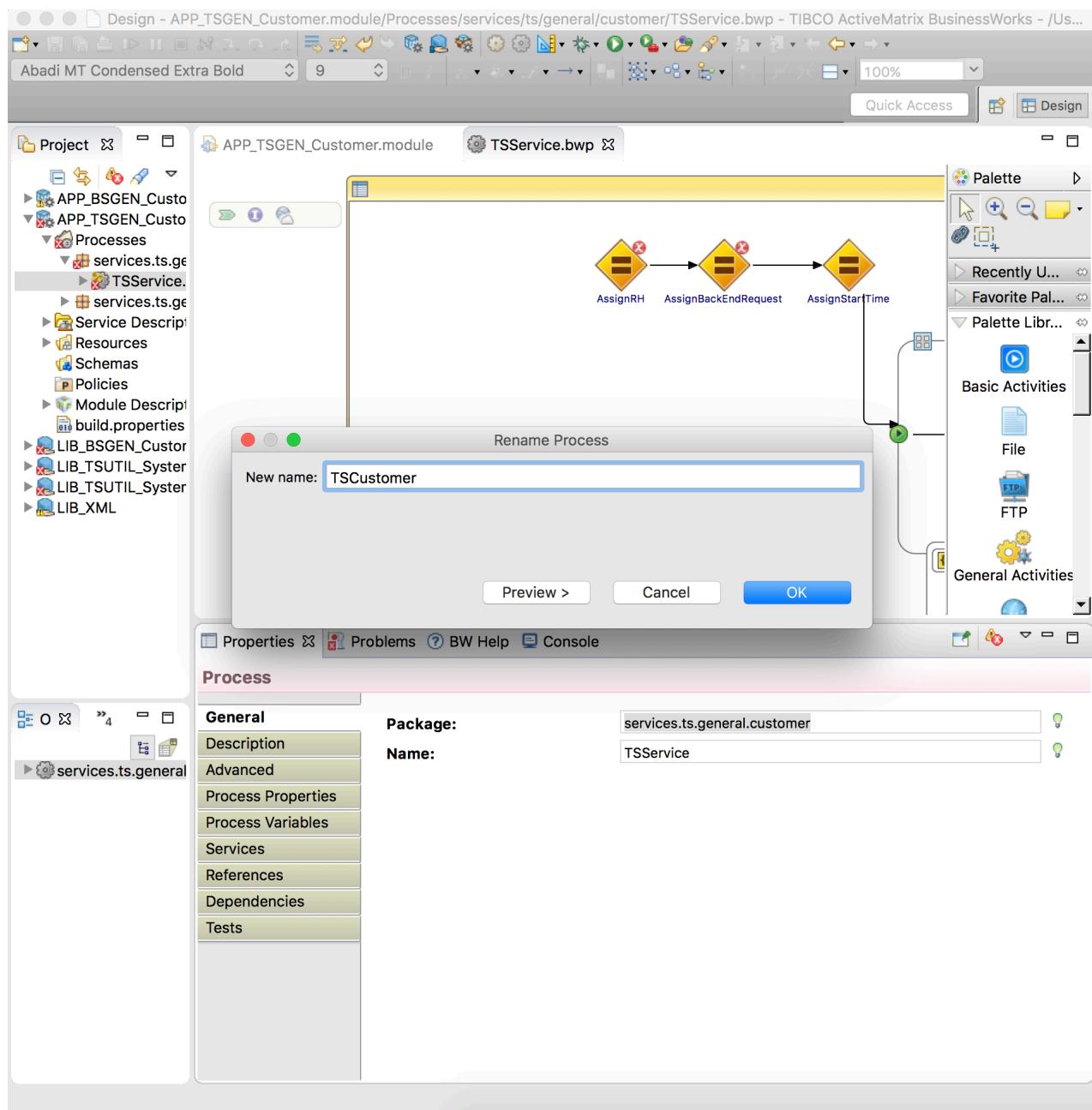


Figure 19: TS Service Provider Shared Module: Process Name

10. Import the XSD and service WSDL of the TS Service into the module.

11. Drag and drop the abstract WSDL into the process:

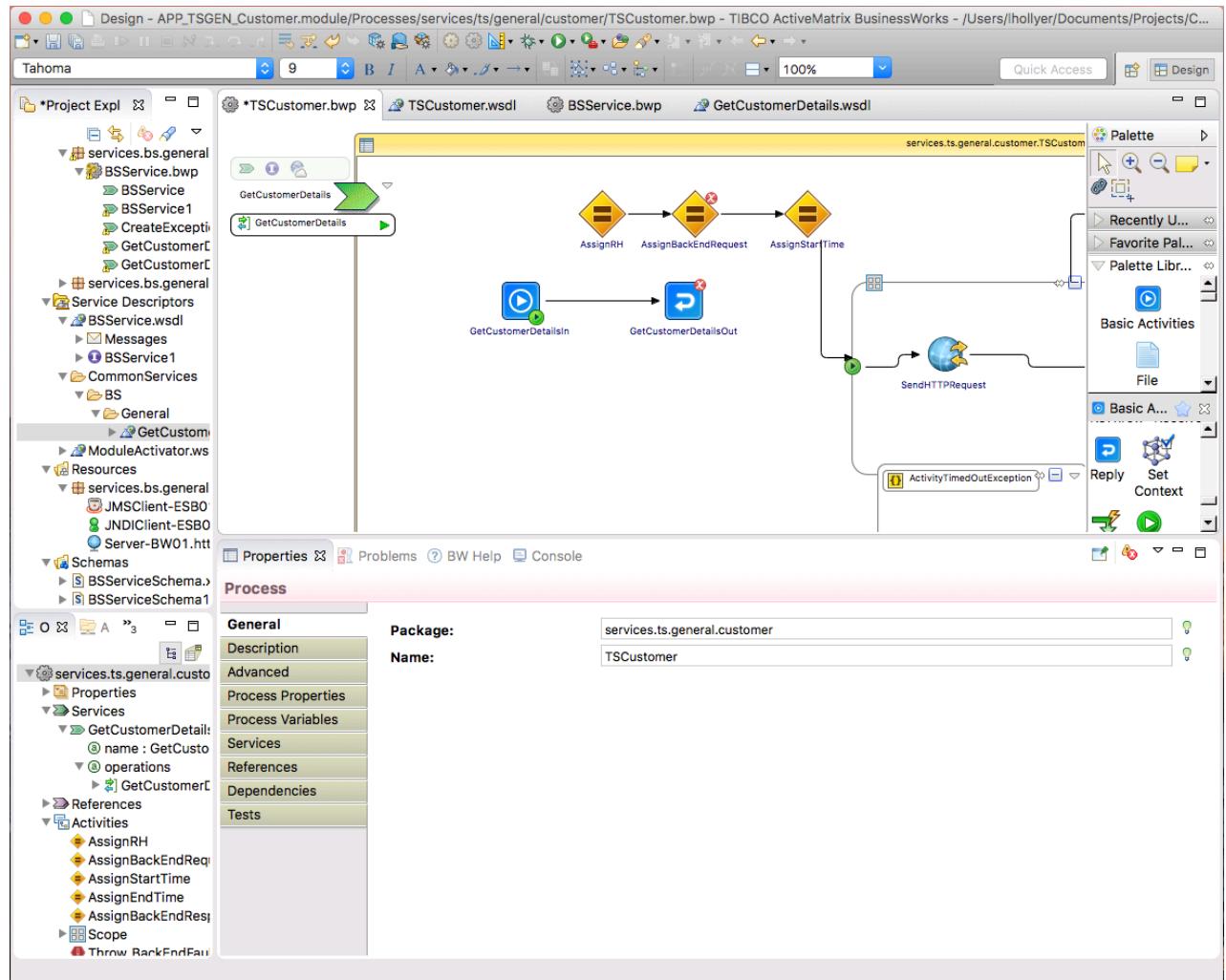


Figure 20: TS Service Provider Shared Module: Implement Service Operation

12. Create a transition between the <Operation>In task and “AssignRH” task:

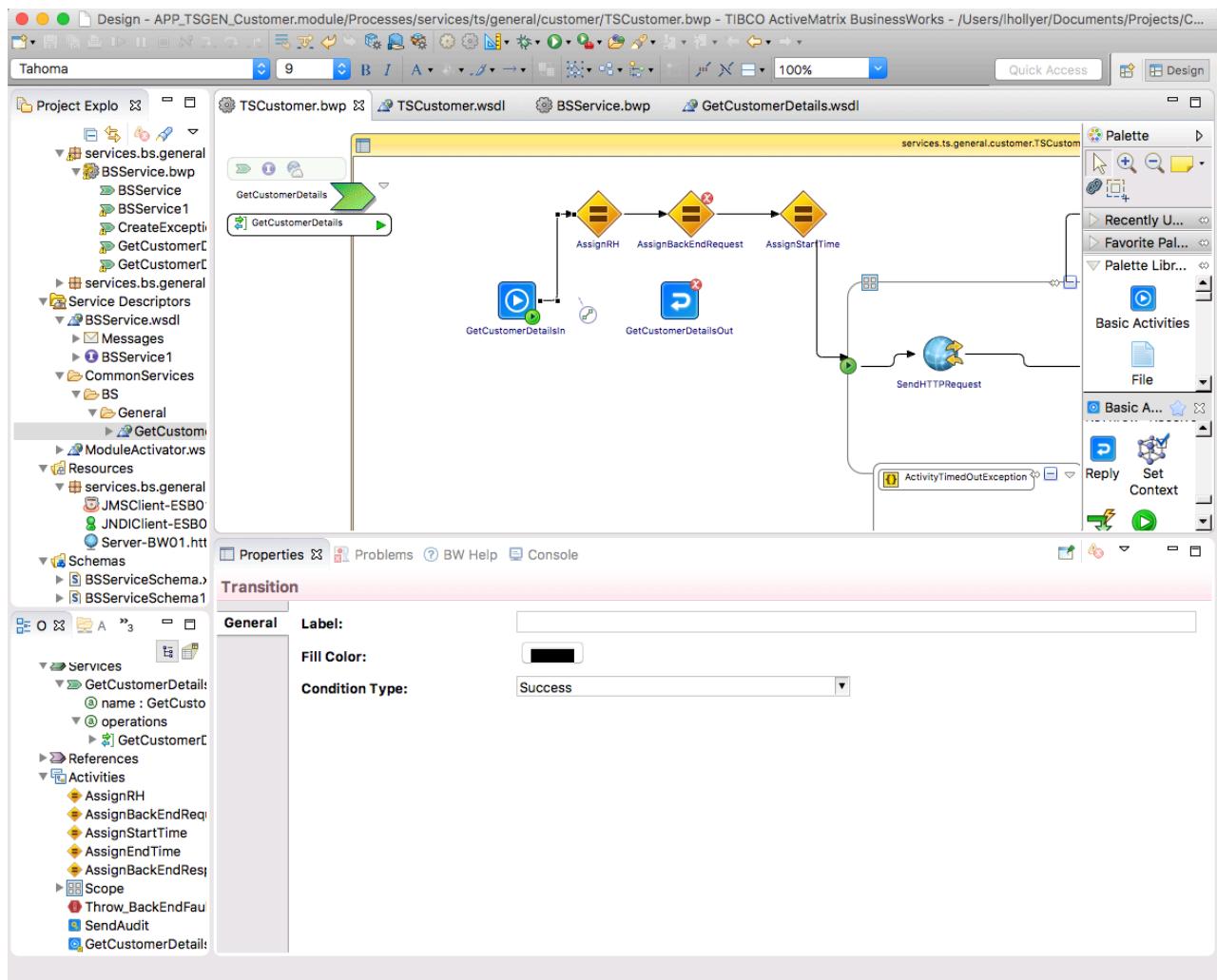


Figure 21: TS Service Provider Shared Module: Change input transition

13. Re-arrange transitions so the <Operation>Out task is located between “AssignBackEndResponse” and “SendAuditEvent” tasks:

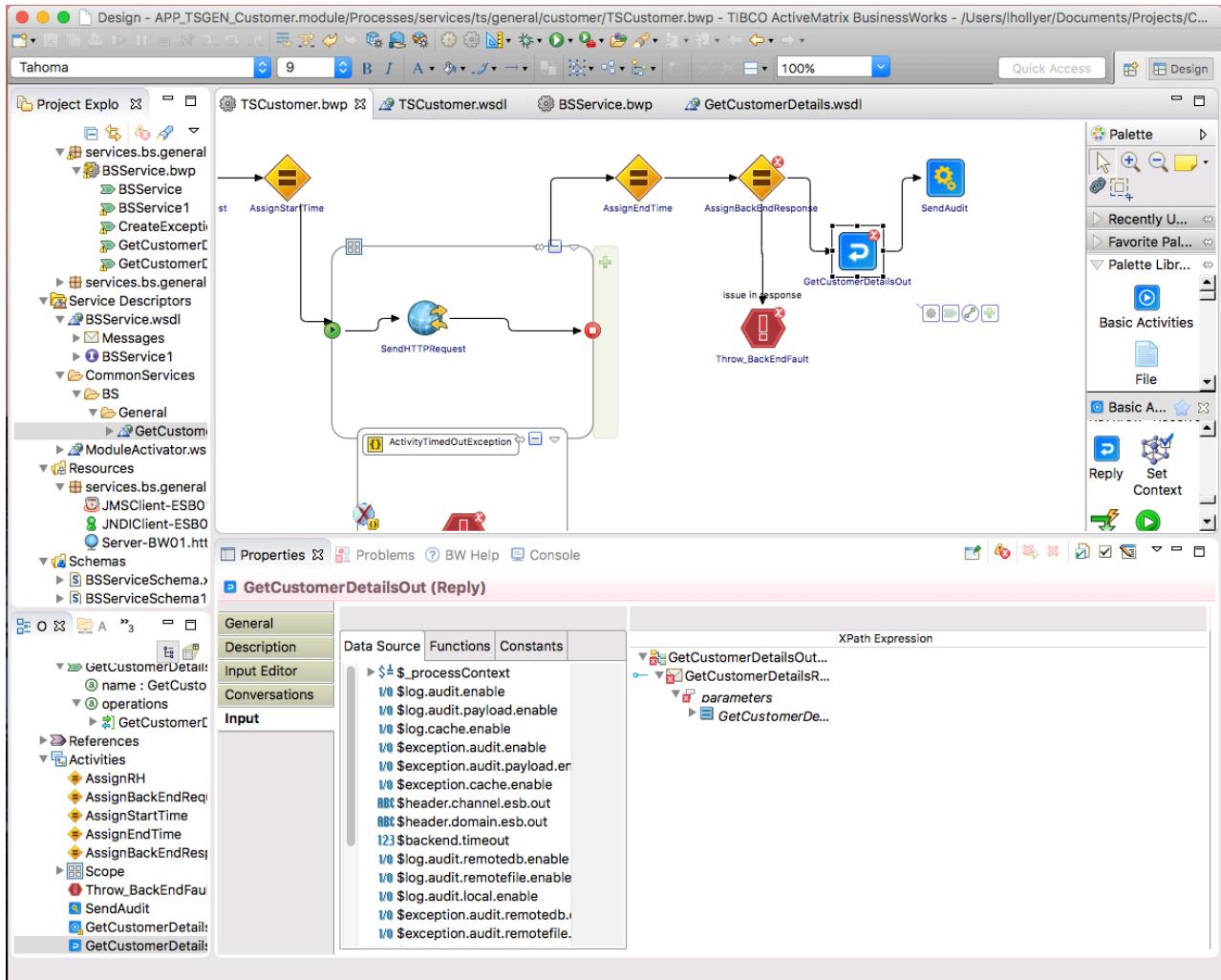


Figure 22: TS Service Provider Shared Module: Reply Transition

14. Save your changes.

## 7.2.5 Fix the Process Mappings

### 7.2.5.1 AssignRH

This task prepares the message header that can be passed into calls to other ESB services or returned in the reply or fault messages.

Fix the mapping in the task “AssignRH” as following:

2. Add a child variable called “varRequest” and change the formula for the “varRequest” variable into

```
$<Operation>In/parameters/tns9:<Operation>Request
```

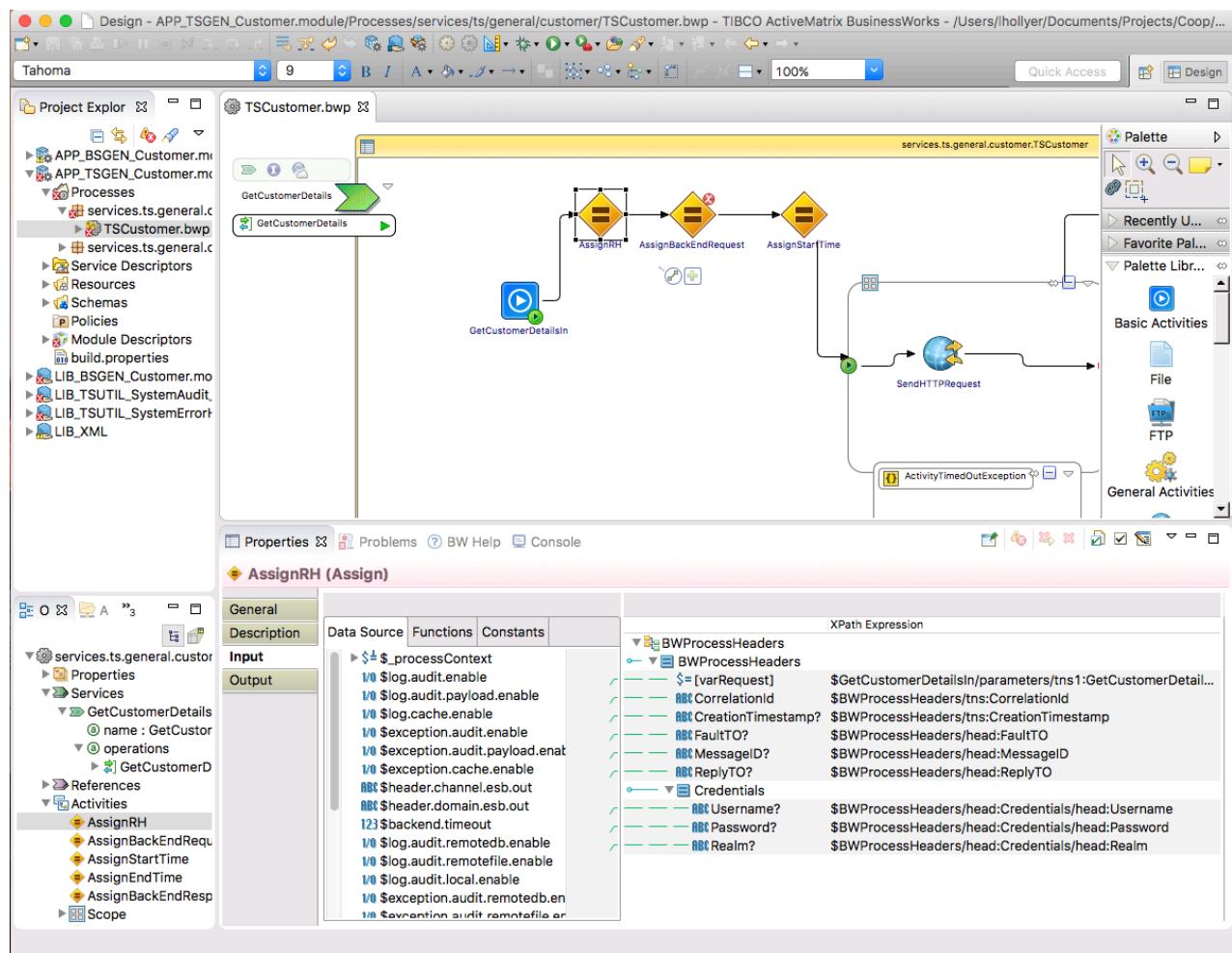


Figure 23: TS Service Provider Shared Module: AssignRH mapping

### 7.2.5.2 Reply\_Fault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML Fault element.

Fix the configuration of the Reply\_Fault task:

2. Select the service, operation and “Fault”.

Fix the mapping of the Reply\_Fault task so:

2. Fault element is a copy of the CreateExceptionEvent / Response / Fault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with no matching condition”.

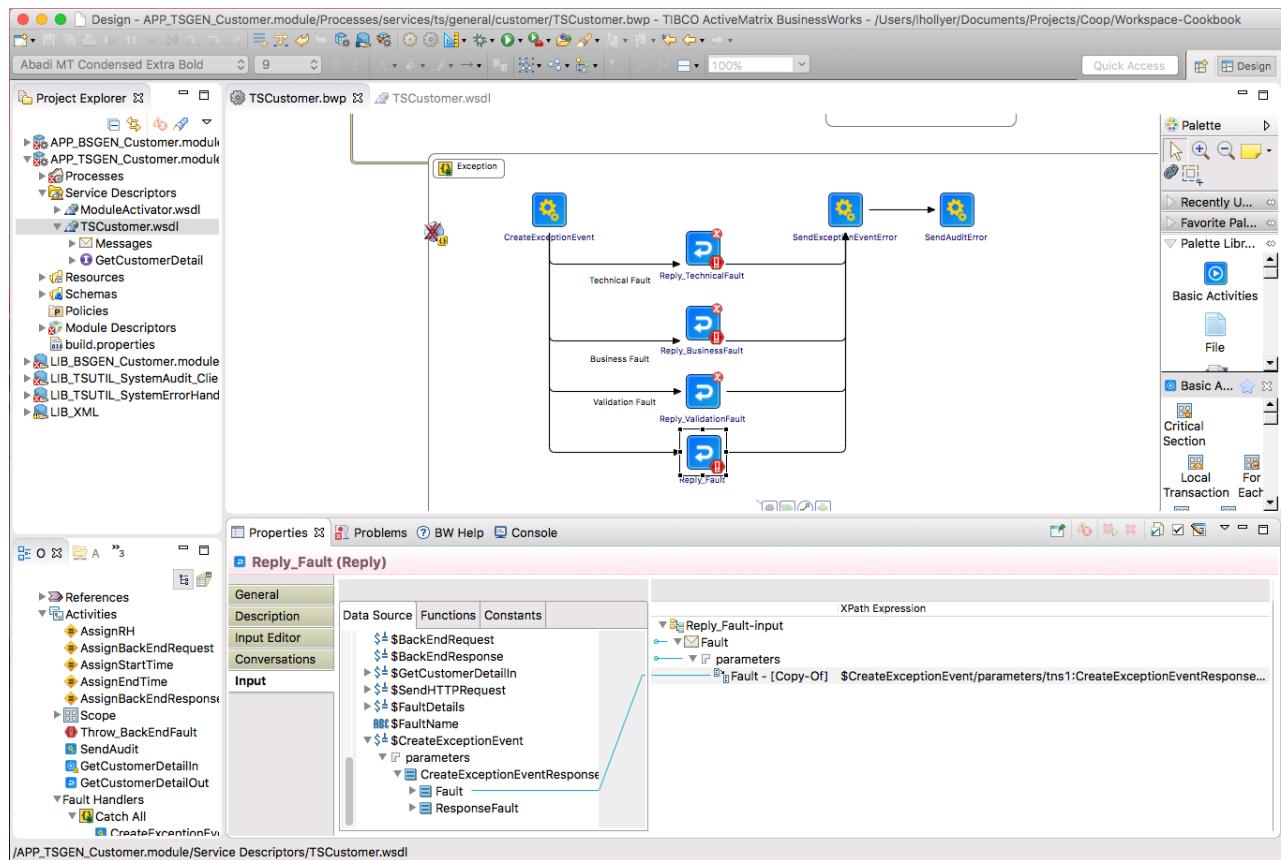


Figure 24: TS Service Provider Shared Module: Reply\_Fault mapping

### **7.2.5.3 Reply BusinessFault**

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML BusinessFault element.

Fix the configuration of the BusinessFault task:

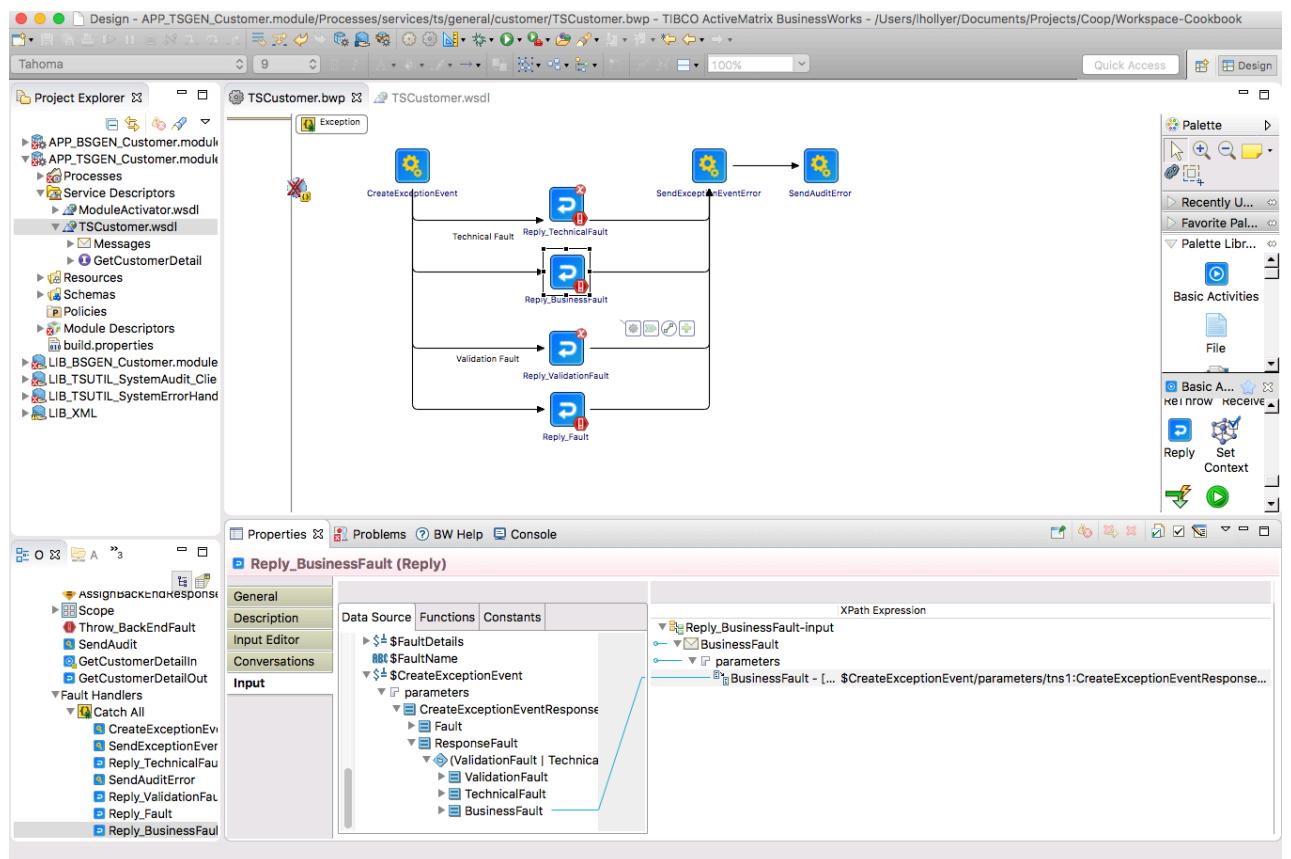
2. Select the service, operation and “BusinessFault”.

Fix the mapping of the BusinessFault task so:

- BusinessFault element is a copy of the CreateExceptionEvent / Response / BusinessFault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent's condition is “Success with matching condition” -

exists(\$CreateExceptionEvent/parameters/tns3>CreateExceptionEventResponse/tns3:ResponseFault/tns2:BusinessFault).



*Figure 25: TS Service Provider Shared Module: Reply\_BusinessFault mapping*

#### 7.2.5.4 Reply\_TechnicalFault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML TechnicalFault element.

Fix the configuration of the TechnicalFault task:

2. Select the service, operation and “TechnicalFault”.

Fix the mapping of the TechnicalFault task so:

2. TechnicalFault element is a copy of the CreateExceptionEvent / Response / TechnicalFault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” –

exists(\$CreateExceptionEvent/parameters/tns3:CreateExceptionEventResponse/tns3:ResponseFault/tns2:TechnicalFault).

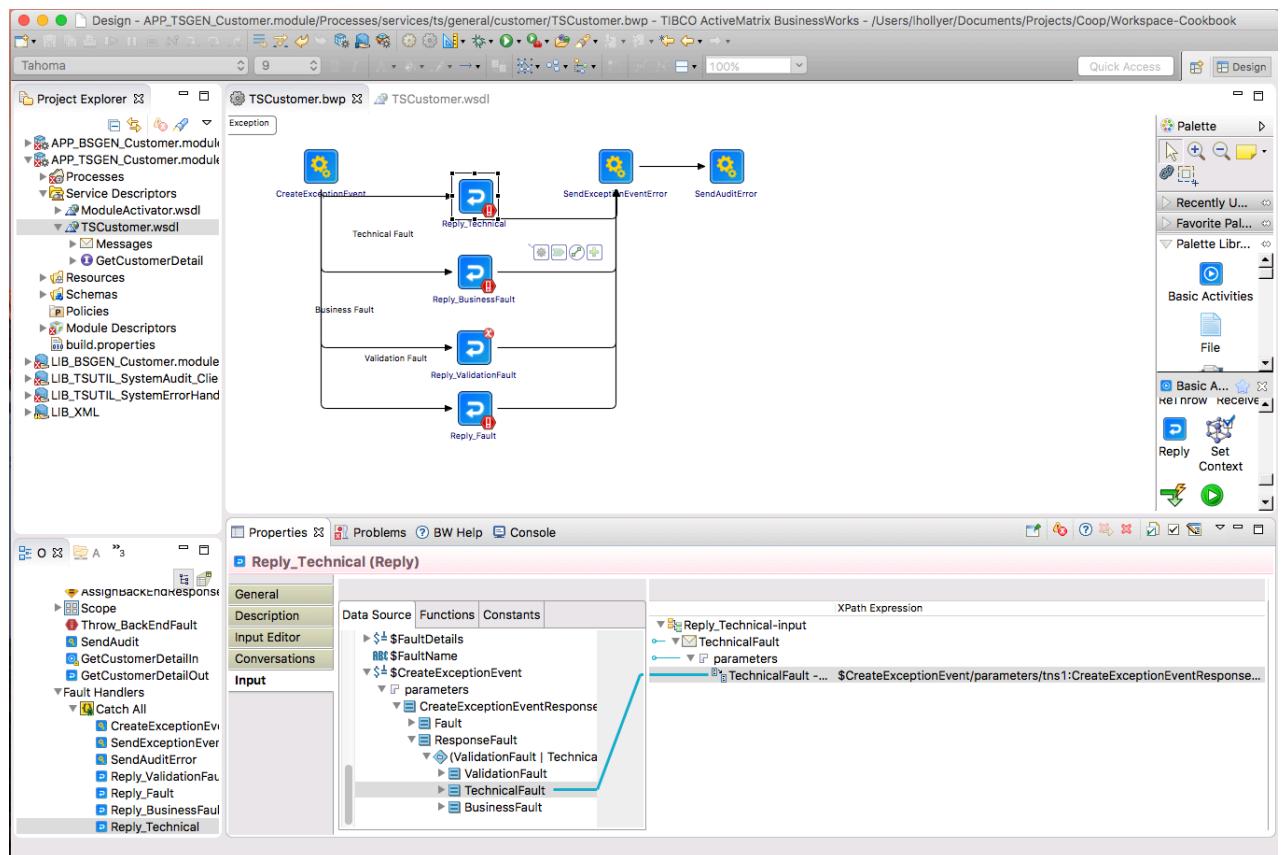


Figure 26: TS Service Provider Shared Module: Reply\_TechnicalFault mapping

### 7.2.5.5 Reply\_ValidationFault

This task returns the fault prepared by CreateExceptionEvent as Fault for the service, in case it contains an XML ValidationFault element.

Fix the configuration of the ValidationFault task:

2. Select the service, operation and “ValidationFault”.

Fix the mapping of the ValidationFault task so:

2. ValidationFault element is a copy of the CreateExceptionEvent / Response / ValidationFault element:

**N.B.** If there are no inputs listed, delete the reply and create a new reply activity and reconnect CreateExceptionEvent and SendExceptionEvent activities – make sure the line from CreateExceptionEvent’s condition is “Success with matching condition” –

exists(\$CreateExceptionEvent/parameters/tns3:CreateExceptionEventResponse/tns3:ResponseFault/tns2:ValidationFault).

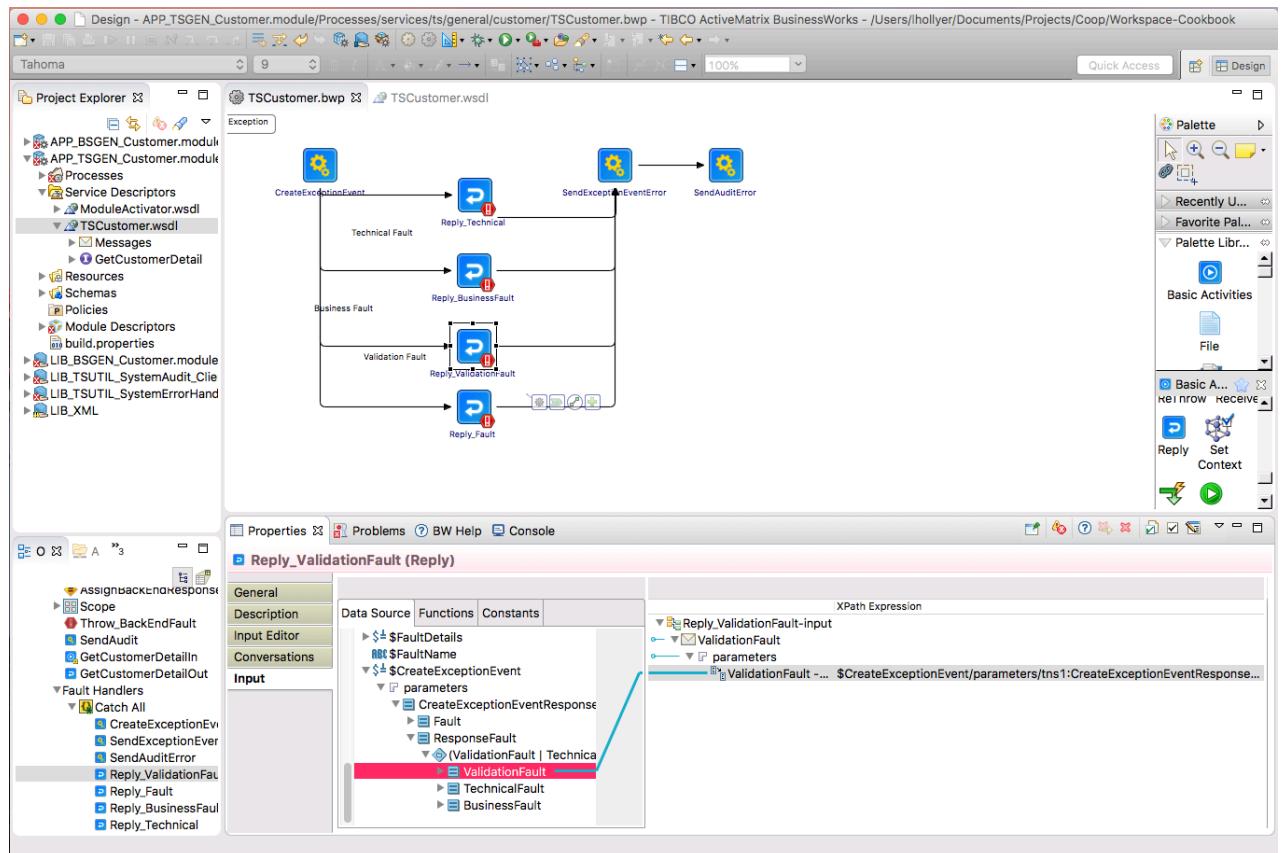


Figure 27: TS Service Provider Shared Module: Reply\_ValidationFault mapping



#### **7.2.5.6 *SendExceptionEvent***

This ask invokes the ESB Audit service to send one event indicating the request that was sent to the back-end and another one indicating the back-end returned a reply.

Fix the mapping of the ValidationFault task as following:

2. Add a “varBackEnd” child variable and change the value of the “varBackEnd” variable so it contains the name of the back-end system, for example “Experian”. If the back-end is .net, please use the .net service name. If the back-end is a DB you can add the name of the stored procedure as well.

#### **7.2.5.7 *SendAuditEvent\_Error***

This ask invokes the ESB Audit service to send one event indicating the request that was sent to the back-end, another one indicating the back-end returned a reply (if it did) and another one indicating an error happened.

Fix the mapping of the ValidationFault task as following:

2. Add a “varBackEnd” child variable and change the value of the “varBackEnd” variable so it contains the name of the back-end system, for example “Experian”. If the back-end is .net, please use the .net service name. If the back-end is a DB you can add the name of the stored procedure as well.

## 7.2.6 Back End on HTTP/HTTPS

If the back-end system is accessed on HTTP/HTTPS transport, you can modify the service as following.

### 7.2.6.1 Change the resources

4. Rename the HTTP Client resource from “services.ts.backend.servicename.Client-BACKEND-http” into “services.ts.<area>.<servicename>.Client-<BackEndName>”.
5. Delete the JDBC resource “Client-BACKEND-jdbc”.
6. Delete the old resource package services.ts.backend.servicename.

### 7.2.6.2 Change the Module Properties

7. Go to the module properties.
8. Select the group “services.ts.<area>.<servicename> / <ServiceName> / backend”. In the properties pane, use the light-bulb icon to rename the group into “<BackEndName>”.

**Example:**

```
services.ts.<area>.<servicename> / <ServiceName> / backend / becomes  
services.ts.bline.getcardscheme / GetCardScheme / BLINE
```

9. In the group you renamed, change the “endpoint.uri” value to correspond to the back-end URI.
10. Select the group “resources / services.ts.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.ts.<area>.<servicename>”.
11. Select the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-http”. In the properties pane, use the light-bulb icon to rename the group into “Client-<BackEndName>”.
12. Delete the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-jdbc”.

### 7.2.6.3 Change Service Process (if back-end is accessed via SOAP)

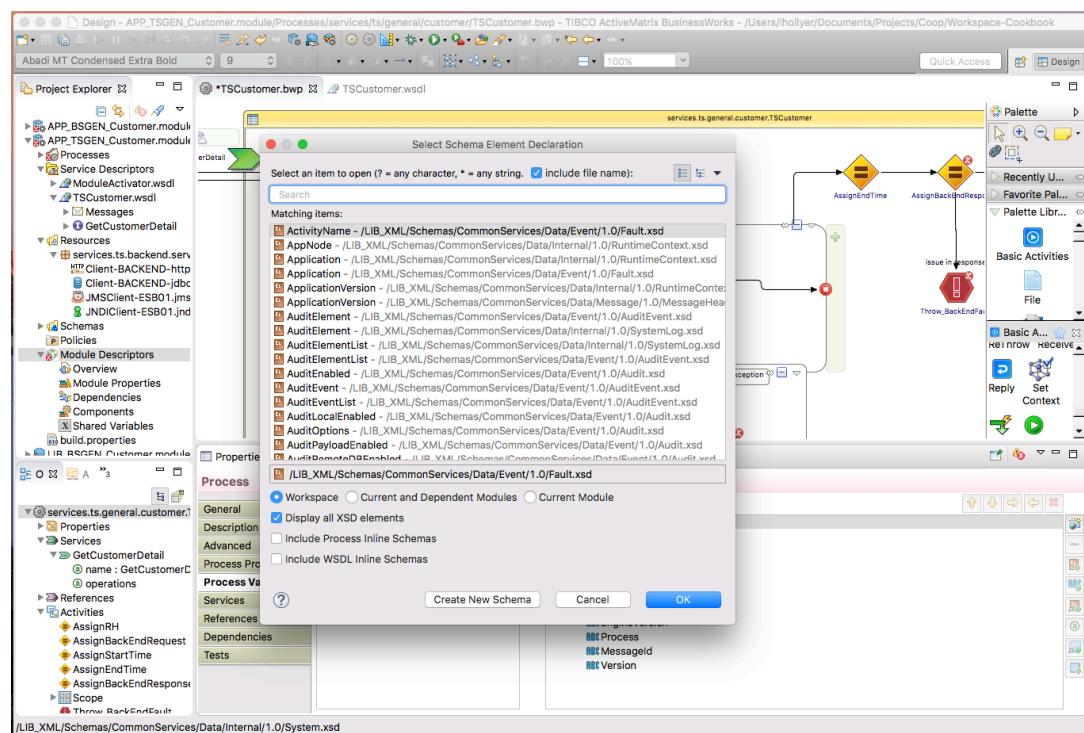
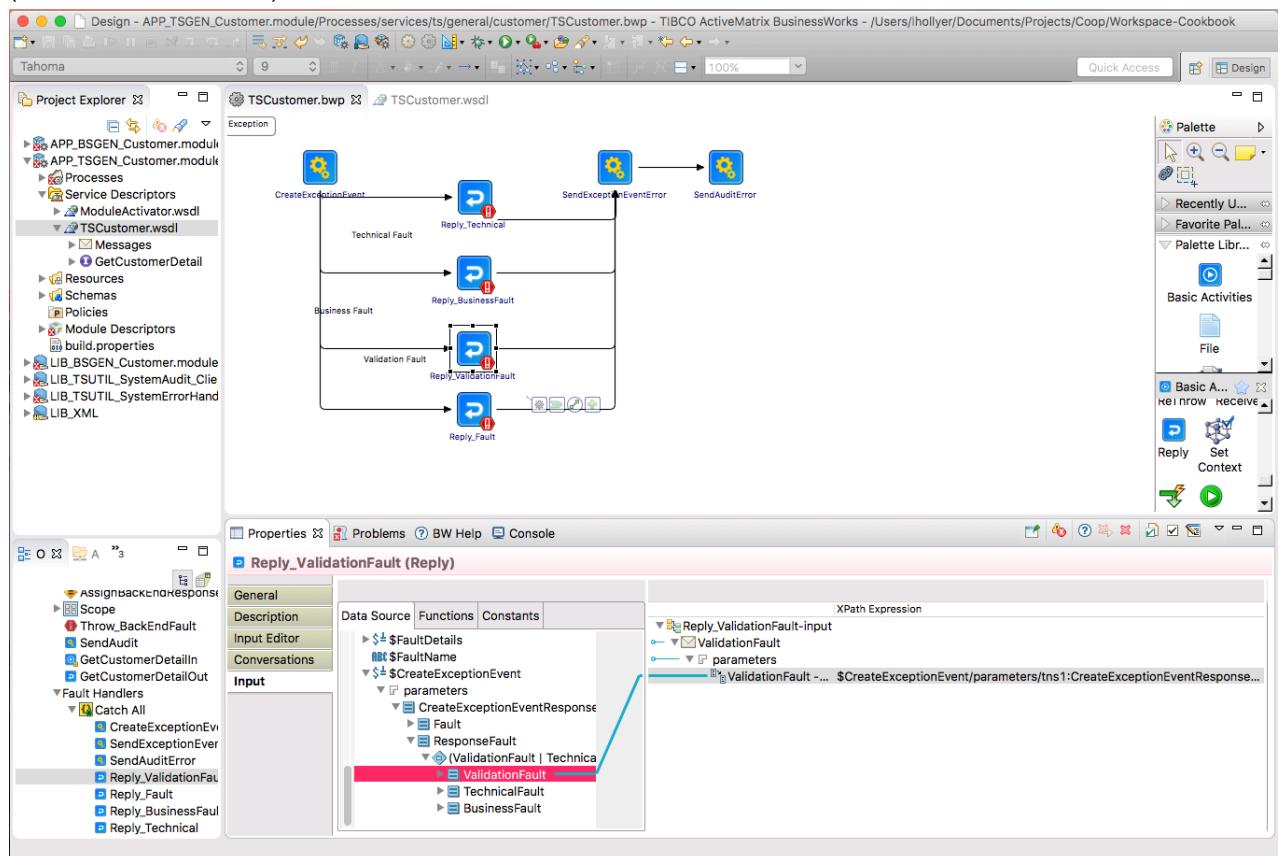
2. Import the concrete WSDL into the “Service Descriptors” folder.

Change the service process as following:

#### Service Invocation

8. Remove the “SendHTTPRequest” task and replace it with an Invoke task.
9. Configure the Service Reference and add a SOAP binding which uses:
  - a. The HTTP client resource “services.ts.<area>.<servicename>.Client-<BackEndName>”.
  - b. The property “services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / endpoint.uri.” for the endpoint URI.

10. Edit the process variable “\$BackEndRequest” so it uses the XML schema of the back-end operation request (from its concrete WSDL).



11. Edit the process variable “\$BackEndResponse” so it uses the XML schema of the back-end operation response (from its concrete WSDL).
12. Map the request to the back-end in the “AssignBackEndRequest”. This task stores the request into the process variable \$BackEndRequest so it can be re-used in other tasks (for the SystemAudit tasks for example).
13. Change the mapping of the Invoke task so it re-uses the \$BackEndRequest variable.
14. Map the response from the back-end in the “AssignBackEndResponse” task. This task stores the response into the process variable \$BackEndResponse so it can be re-used in other tasks (for the SystemAudit tasks for example).

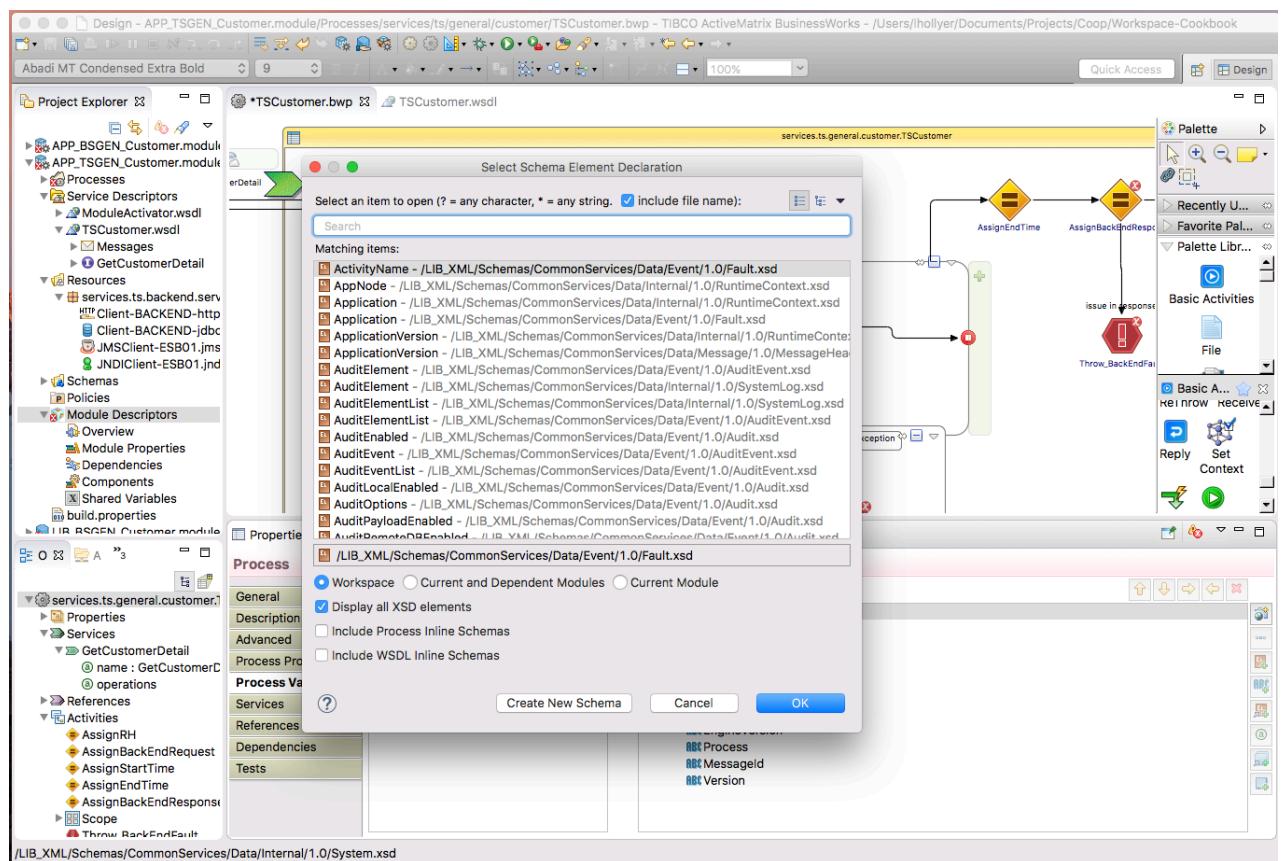
#### **7.2.6.4 Change Service Process (*if back-end is accessed via raw HTTP*)**

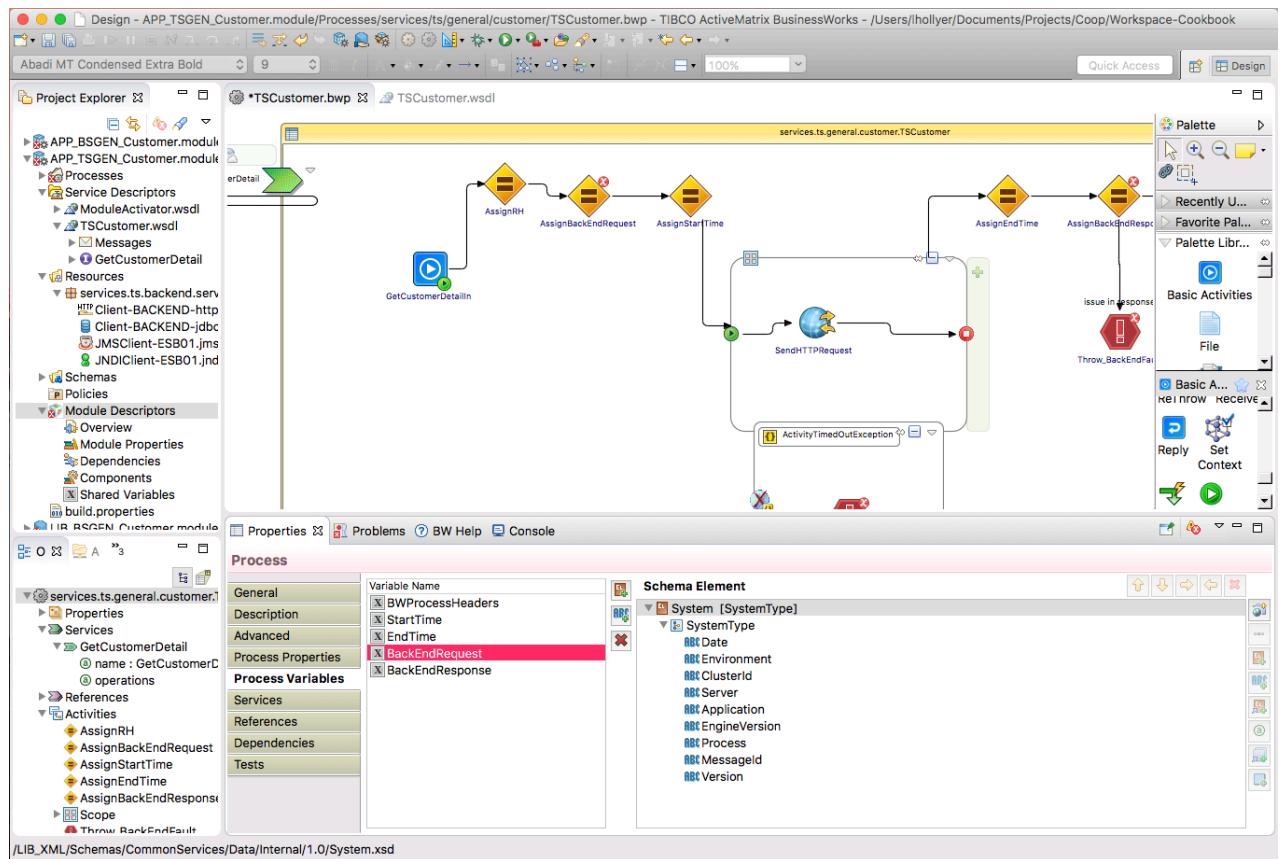
Change the service process as following:

##### **Service Invocation**

6. Edit the process variable “\$BackEndRequest” so it uses the XML schema of the back-end operation request.

## CO-OP Bank - ESB – How To: Create BW Services from Templates





7. Edit the process variable “\$BackEndResponse” so it uses the element “Payload” from the **LIB\_XML / Schemas / CommonServices / Data / Event / 1.0 / AuditEvent.xsd** XSD.
8. Map the request to the back-end in the “AssignBackEndRequest”. This task stores the request into the process variable \$BackEndRequest so it can be re-used in other tasks (for the SystemAudit tasks for example).
9. Change the mapping of the Invoke task so it re-uses the \$BackEndRequest variable.
10. Map the \$SendHTTPRequest variable into the input the “AssignBackEndResponse” task. This task stores the response into the process variable \$BackEndResponse so it can be re-used in other tasks (for the SystemAudit tasks for example).

#### **7.2.6.5 Change Service Process (all cases)**

##### **Detect Back End Errors**

If the service must analyze the back-end reply to detect back-end errors, this can be done in the “Throw\_BusinessFault” task and the transition from “AssignBackEndResponse” task to “Throw\_BusinessFault” task. In the “Throw\_BusinessFault” task you can change the text and description of the issue.

If not, you can delete the “Throw\_BusinessFault” task.

If the service operation can throw various SOAP faults that can be mapped to an ESB BusinessFault, you can handle them as following:

9. Right-click on the Invoke task and in the contextual menu, select the option "Catch / <fault>" where <fault> is the SOAP fault to handle.
  10. In the new Catch block, add a "Throw" task.
  11. In "Input Editor" of the "Throw" task, add the XML element "ThrowableBusinessFault" from the "Schemas / CommonServices / Data / Message / Internal / Throwable" XSD.
  12. Change the mapping of the task to populate the text and description of the fault.

## Timeout Faults

4. In the “Throw\_TimeoutFault” task, you can change the text and description of the timeout error.

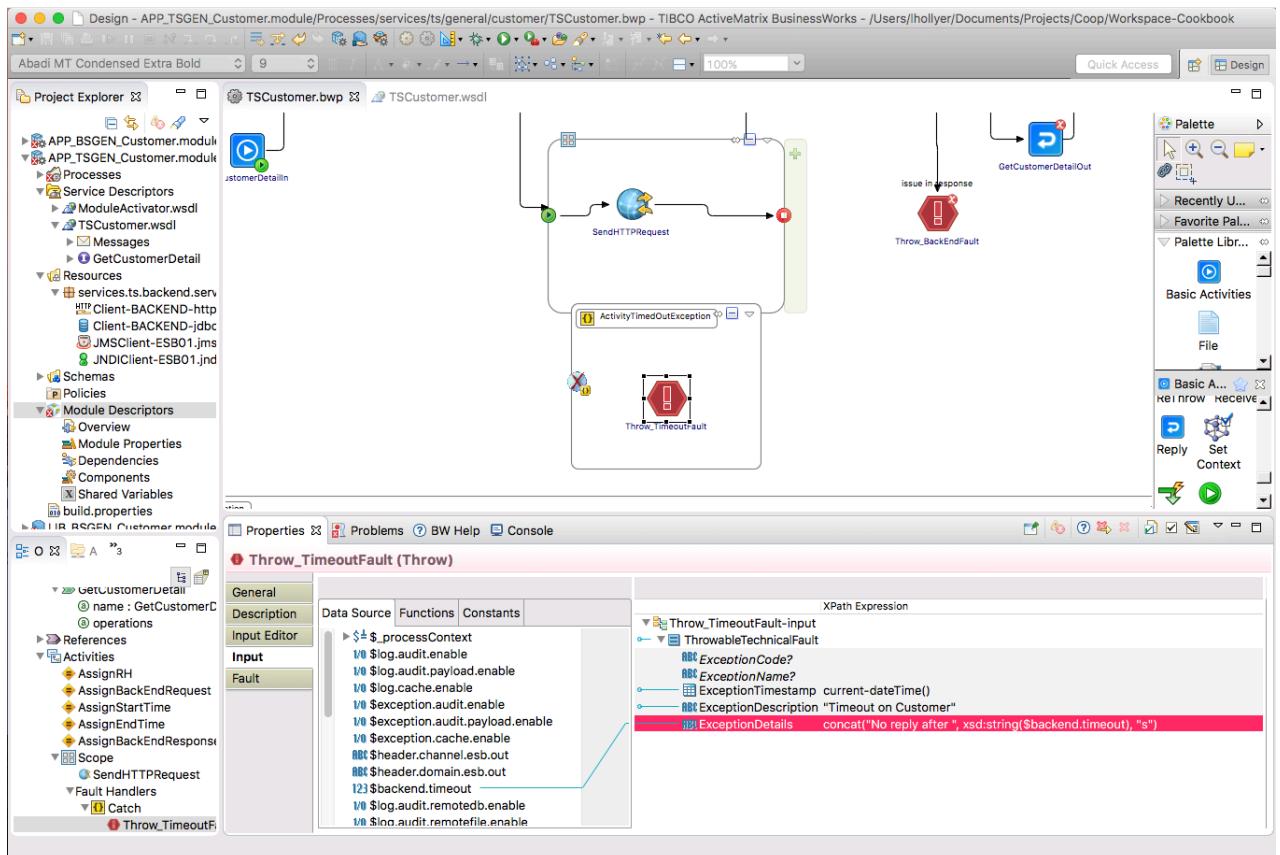


Figure 28: TS Service Provider Shared Module: Throw\_TimeoutFault mapping

5. If the task has no input, fix it as following: in “Input Editor” add the XML element “ThrowableBusinessFault” from the “Schemas / CommonServices / Data / Internal / 1.0 / Internal / Throwable” XSD:

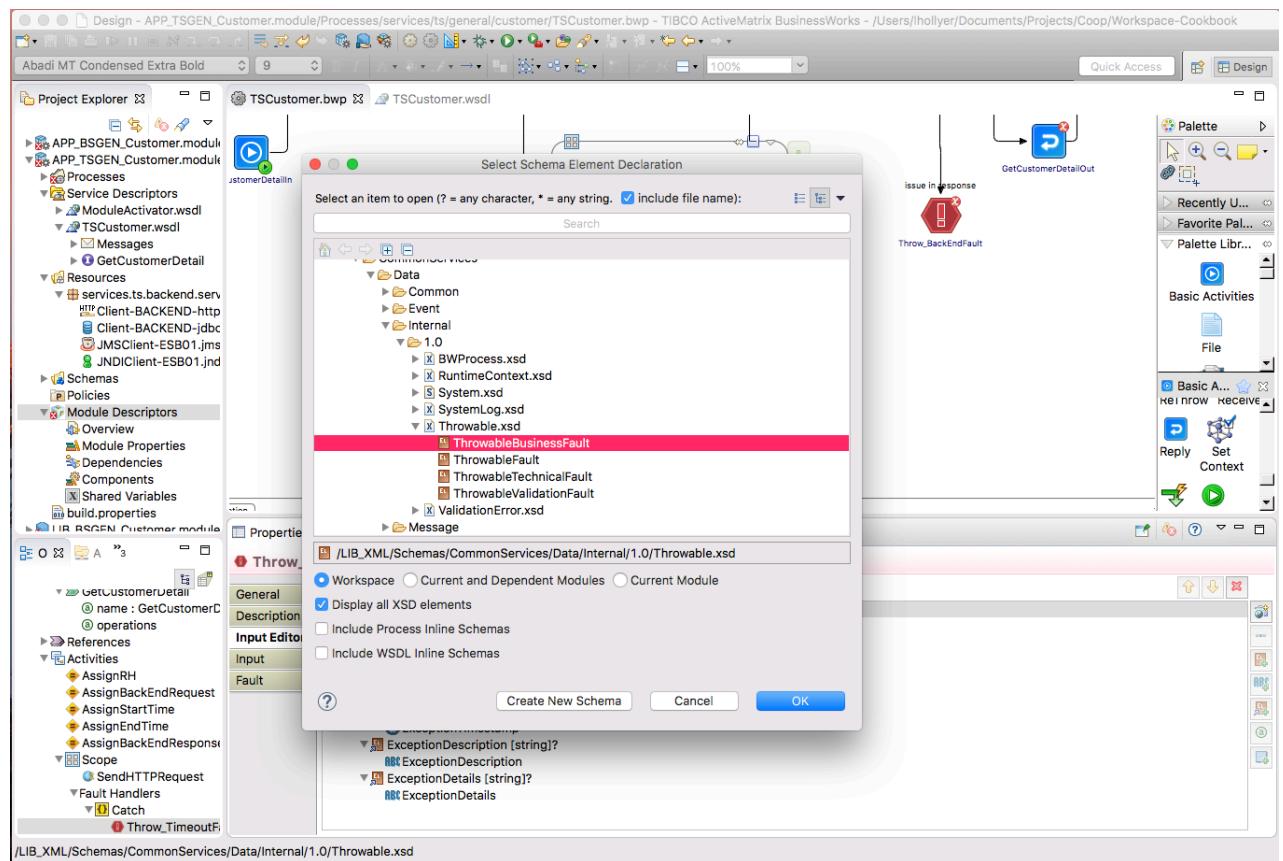


Figure 29: TS Service Provider Shared Module: Throw\_TimeoutFault configuration

6. Delete and recreate the task if necessary.

#### 7.2.6.6 Add HTTPS Resources

If the back-end is accessed via HTTPS, the following resources must be added manually and named as such:

- Resources:
  - SSL Client Configuration: `services.ts.<area>.<servicename>.Client-<BackEndName>-ssl`
  - Keystore Provider Resource: `services.ts.<area>.<servicename>.Client-<BackEndName>-jks`
- Module Properties
  - Keystore URL: `services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / keystore.url.`
  - Keystore Password: `services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / keystore.password.`
  - Refresh Interval: `services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / keystore.refresh.interval` (3600000 ms as default value)

### 7.2.7 Back End on JDBC

If the back-end system is a database, you can modify the service as following.

#### 7.2.7.1 Change the resources

4. Rename the JDBC Client resource from “services.ts.backend.servicename.Client-BACKEND-jdbc” into “services.ts.<area>.<servicename>.Client-<BackEndName>”.
5. Delete the HTTP resource “Client-BACKEND-http”.
6. Delete the old resource package services.ts.backend.servicename.

#### 7.2.7.2 Change the Module Properties

7. Go to the module properties.
8. Select the group “services.ts.<area>.<servicename> / <ServiceName> / backend / BACKEND”. In the properties pane, use the light-bulb icon to rename the group into “<BackEndName>”.
9. In the group you renamed, delete the “endpoint.uri” property.
10. Select the group “resources / services.ts.backend.servicename”. In the properties pane, use the light-bulb icon to rename the group into “services.ts.<area>.<servicename>”.
11. Select the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-jdbc”. In the properties pane, use the light-bulb icon to rename the group into “Client-<BackEndName>”.
12. Delete the group “resources / services.ts.<area>.<servicename> / Client-BACKEND-http”.

#### 7.2.7.3 Change Service Process

Change the service process as following:

7. Remove the “SendHTTPRequest” task and replace it with a JDBC task.
8. Configure the JDBC tasks so it uses:
  - a. The JDBC client resource “services.ts.<area>.<servicename>.Client-<BackEndName>”.
  - b. The property “services.ts.<area>.<servicename> / <ServiceName> / backend / <BackEndName> / timeout” for the timeout.
9. Map the request to the back-end in the “AssignBackEndRequest”. This task stores the request into the process variable \$BackEndRequest so it can be re-used in other tasks (for the SystemAudit tasks for example).
10. Map the response from the back-end in the “AssignBackEndResponse” task. This task stores the response into the process variable \$BackEndResponse so it can be re-used in other tasks (for the SystemAudit tasks for example).
11. If the service must analyze the back-end reply to detect back-end errors, this can be done in the “Throw\_BusinessFault” task and the transition from “AssignBackEndResponse” task to “Throw\_BusinessFault” task. In the “Throw\_BusinessFault” task you can change the text and description of the issue.

If not, you can delete the “Throw\_BusinessFault” task.

12. In the “Throw\_TimeoutFault” task, you can change the text and description of the timeout error.

## 7.3 Implement the Service

### 7.3.1 Service Response

In the <Operation>Out task, you MUST map the ResponseHeader element as following:

- **CorrelationId:** \$BWProcessHeaders / CorrelationId
- **MessageID:** \$BWProcessHeaders / MessageID
- **Version:** \$BWProcessHeaders / Response / Version
- **ResponseDateTime:** current-dateTime()

**N.B.** if <Operation>Out does not have the ResponseHeader element, check the WSDL location imports the Schema in APP\_TS<Service>.module's XSD, and check the XSD has the ResponseHeader set to ResponseHeaderType from LIB\_XML.

### 7.3.2 Invoking another ESB service

If you invoke another ESB service (BS or TS), you MUST map the RequestHeader element as following:

- **CorrelationId:** \$BWProcessHeaders / CorrelationId
- **MessageID:** \$BWProcessHeaders / MessageID
- **Version:** The version of the invoked service, for example “1.0”.
- **RequestDateTime:** current-dateTime()

**N.B.** if <Operation>Out does not have the ResponseHeader element, check the WSDL location imports the Schema in APP\_TS<Service>.module's XSD, and check the XSD has the RequestHeader set to RequestHeaderType from LIB\_XML.

### 7.3.3 Validate the Request

Some service may require additional input data validation (besides what is defined inside the service schema, which BW validates by default). In such case, you must implement the validation as following:

- The steps are the same as for a business service (see section 4.3.5), but the “Throw\_ValidationFault” happens between the “AssignRH” task and “AssignBackEndRequest” task.

### 7.3.4 Module Properties, Processes, Service Invocation

The rest of the procedure is the same as for a technical service provider in a Shared Module, please refer to the previous chapter for all required procedure information, except for the next sections, which are specific to an application.

### 7.3.5 Using JMS Queue

- Bind the JMS connection to the **south** EMS server.

- TSUTIL\_SystemAudit\_Client and TSUTIL\_SystemErrorHandler\_Client use their own JMS connection resources so deleting the resources from the Business service is not an issue. They are the only services to bind to **logging** EMS server.

### 7.3.6 Create the SOAP Binding

Once your service process has been renamed and moved to the proper Process Package, you must create its SOAP over JMS binding.

1. Expand “Module Properties” in the module Project Explorer.
2. Double-click on “Module Descriptors / Components”:

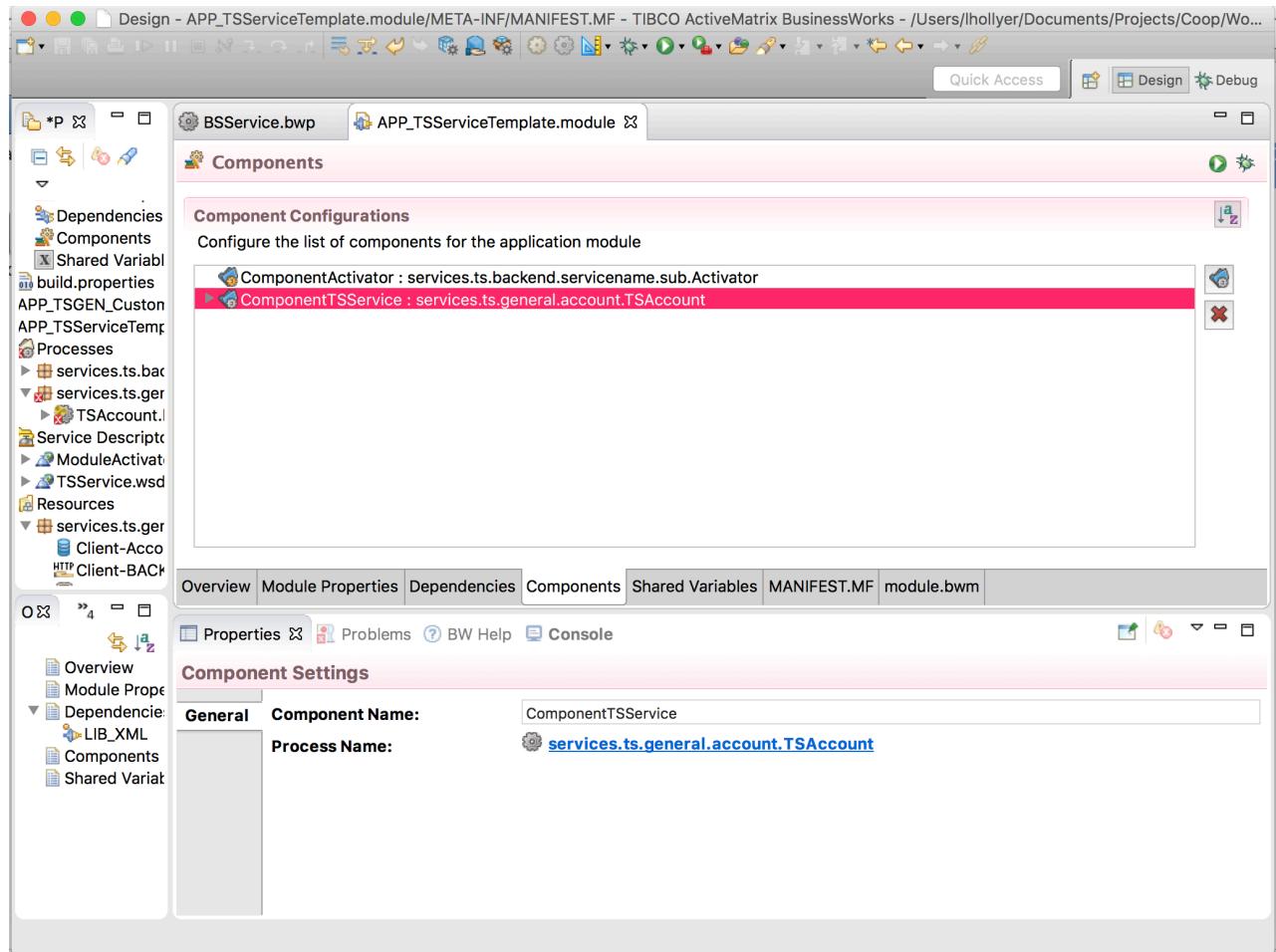


Figure 30: Create the SOAP over JMS Interface: Module Components

3. Click on the arrow on the left of “Component<serviceName>”, select “<serviceName>” and select “Properties” in the pane below (you may to click several times on <serviceName> to see the properties appear as below):

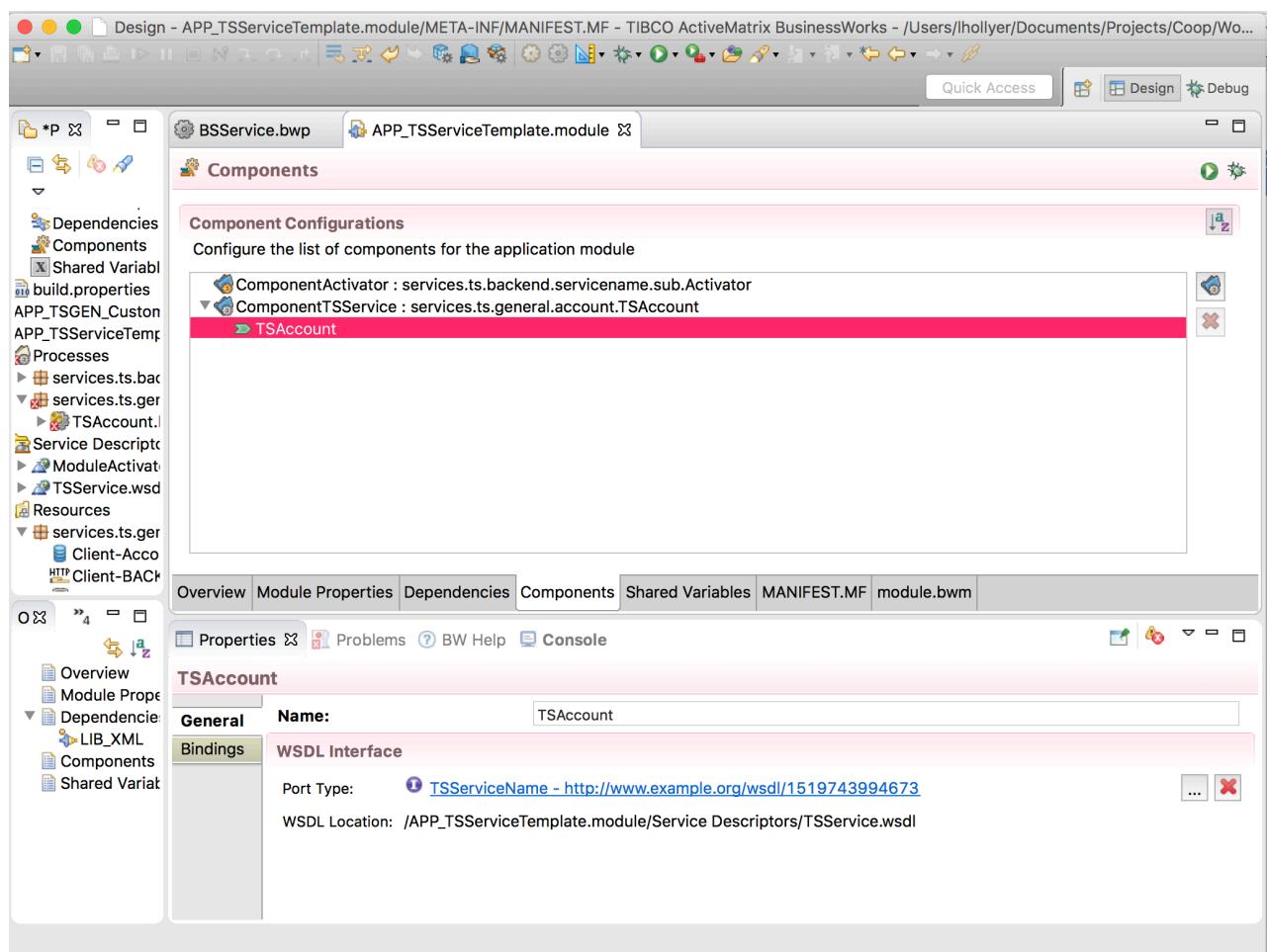


Figure 31: Create the SOAP over JMS Interface: Service Settings

4. Select “Bindings” and click on the “+” button.
5. In the “Add Binding” dialog box:
  - a. Select “SOAP Binding”,
  - b. Check the box “Select / Create Required Resources”.

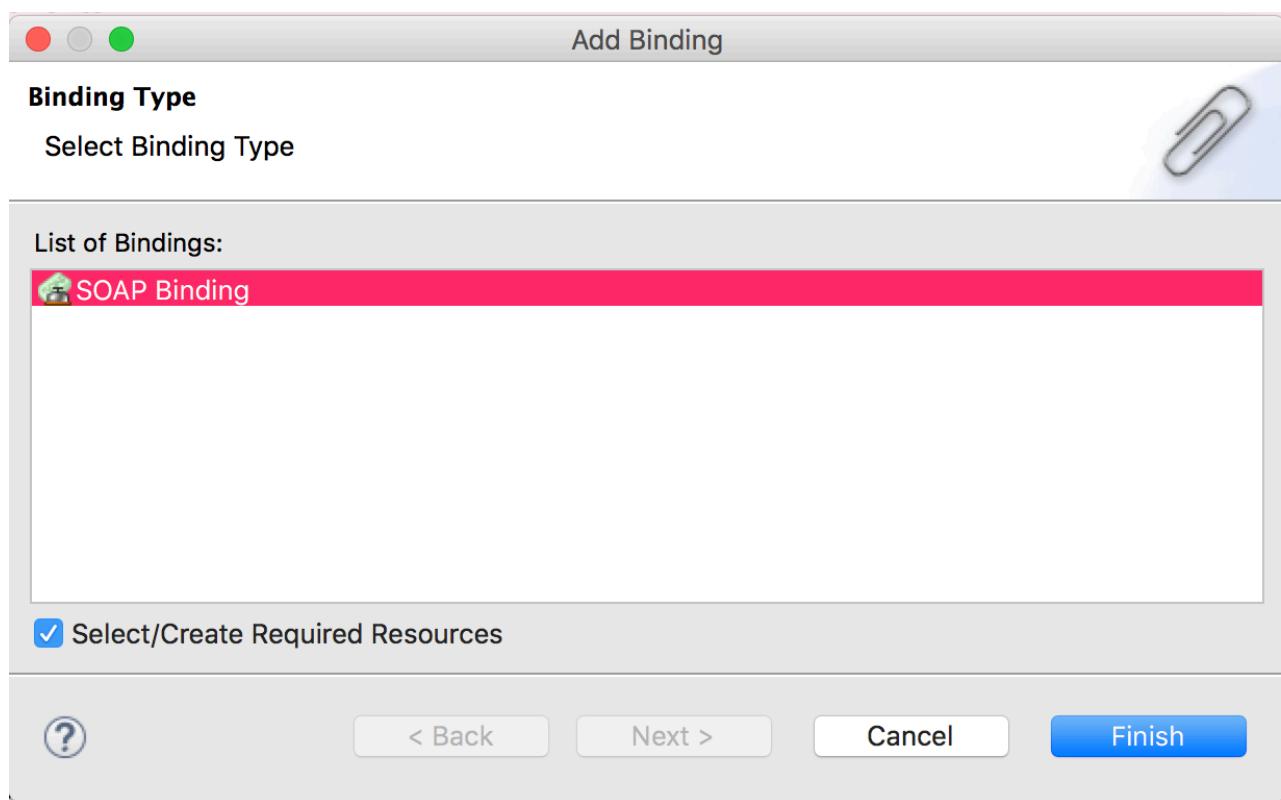


Figure 32: Create the SOAP over JMS Interface: New SOAP Binding

6. Click on the “Finish” button.

## CO-OP Bank - ESB – How To: Create BW Services from Templates

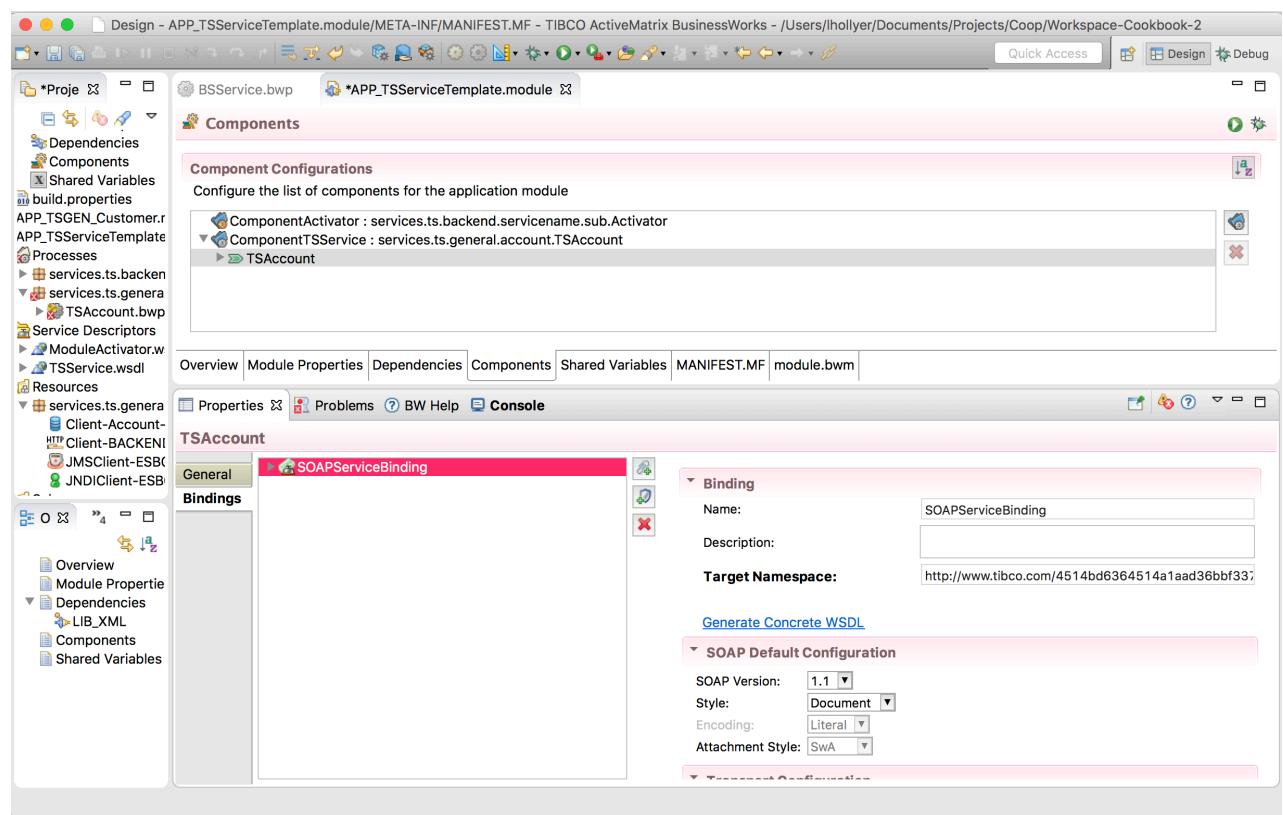


Figure 33: Create the SOAP over JMS Interface: New SOAP Binding

7. In the new “Binding” panel that appeared:

- Select “JMS” in the “Transport Type”.

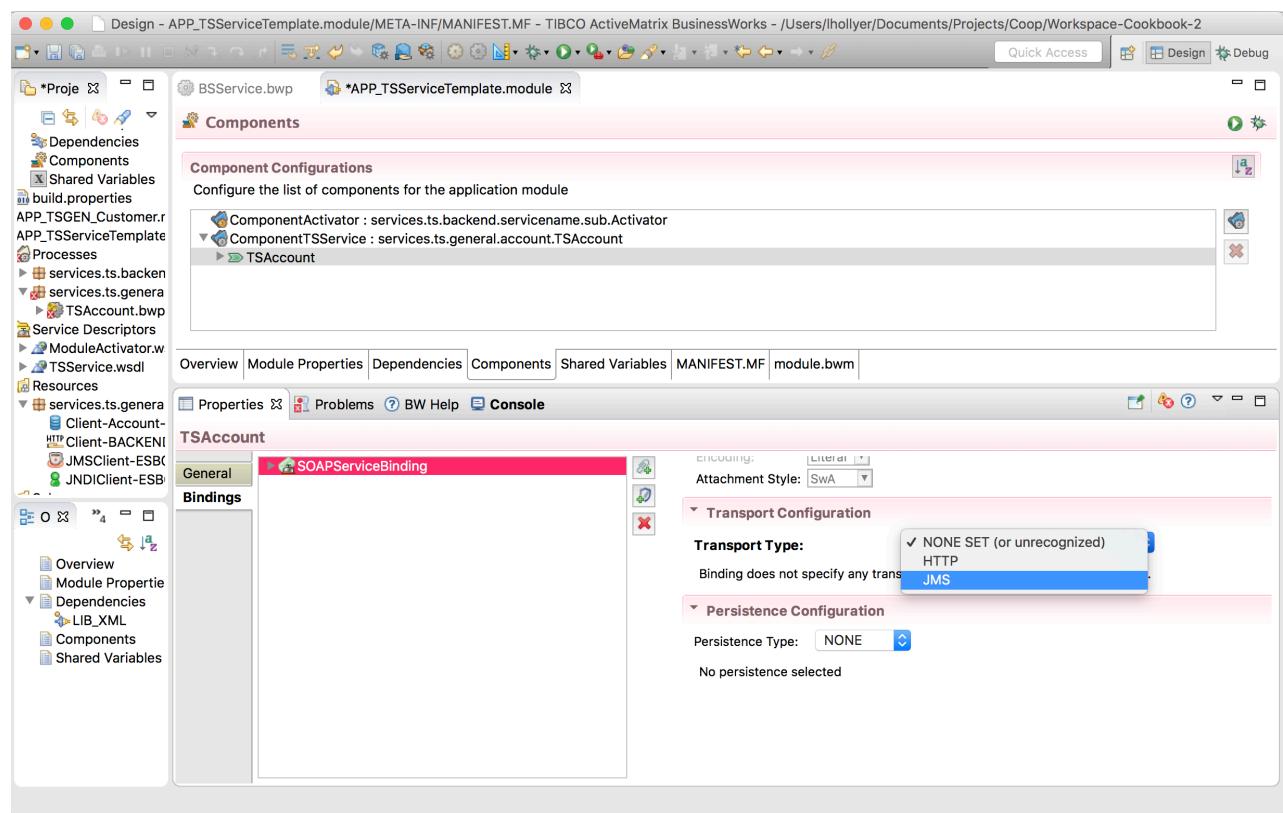
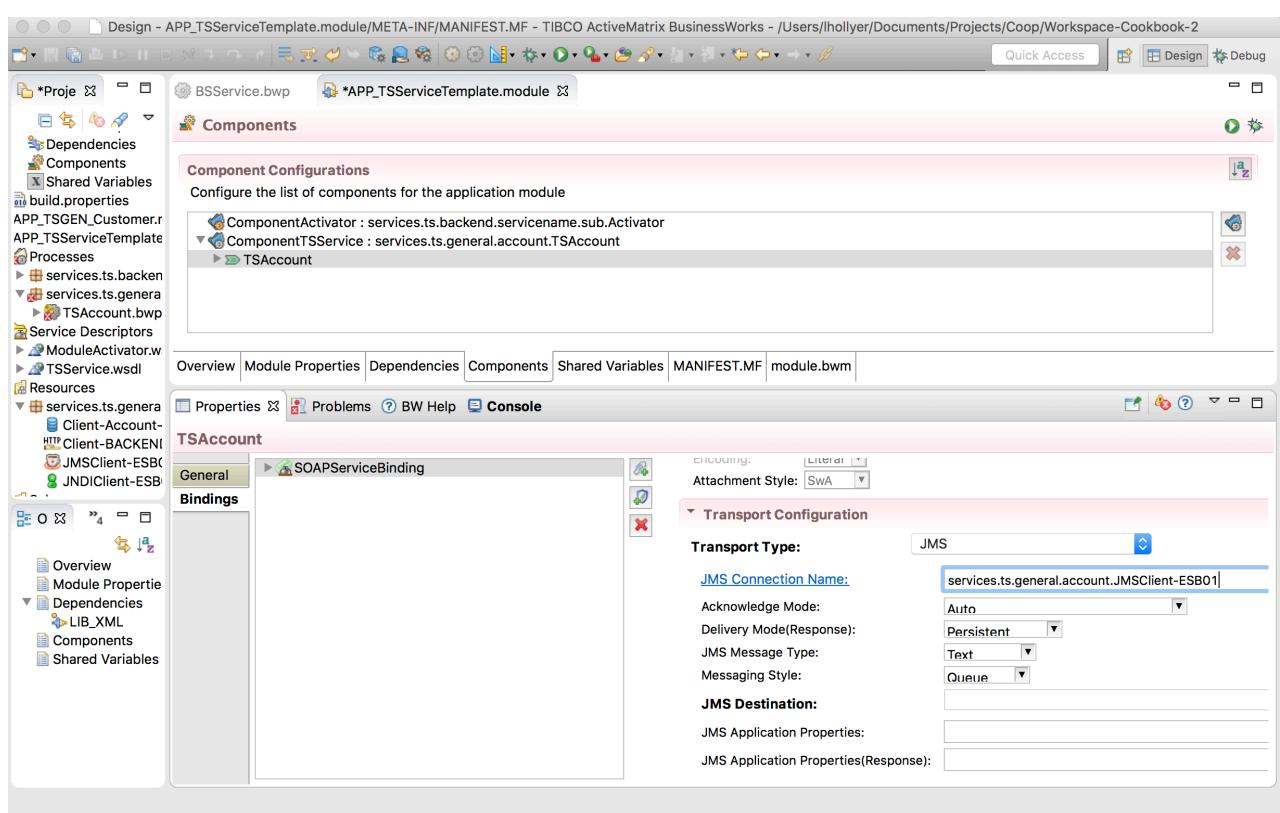


Figure 34: Create the SOAP over JMS Interface: Transport Type

8. In the “Select Jms... Dialog” box, select the JMS connection from the current module (the JMS Client resource renamed in previous steps)/
9. Now, back into the “Binding” panel:
  - b. Select Acknowledgment Mode as “Auto”,
  - c. Select JMS Message Type as “Text”,
  - d. Select Messaging Style as “Queue”.

## CO-OP Bank - ESB – How To: Create BW Services from Templates



*Figure 35: Create the SOAP over JMS Interface: JMS Settings*

10. Change the JMS Destination so it uses the module property `services.ts.<area>.<servicename> / <ServiceName> / jms.queue:`

## CO-OP Bank - ESB – How To: Create BW Services from Templates

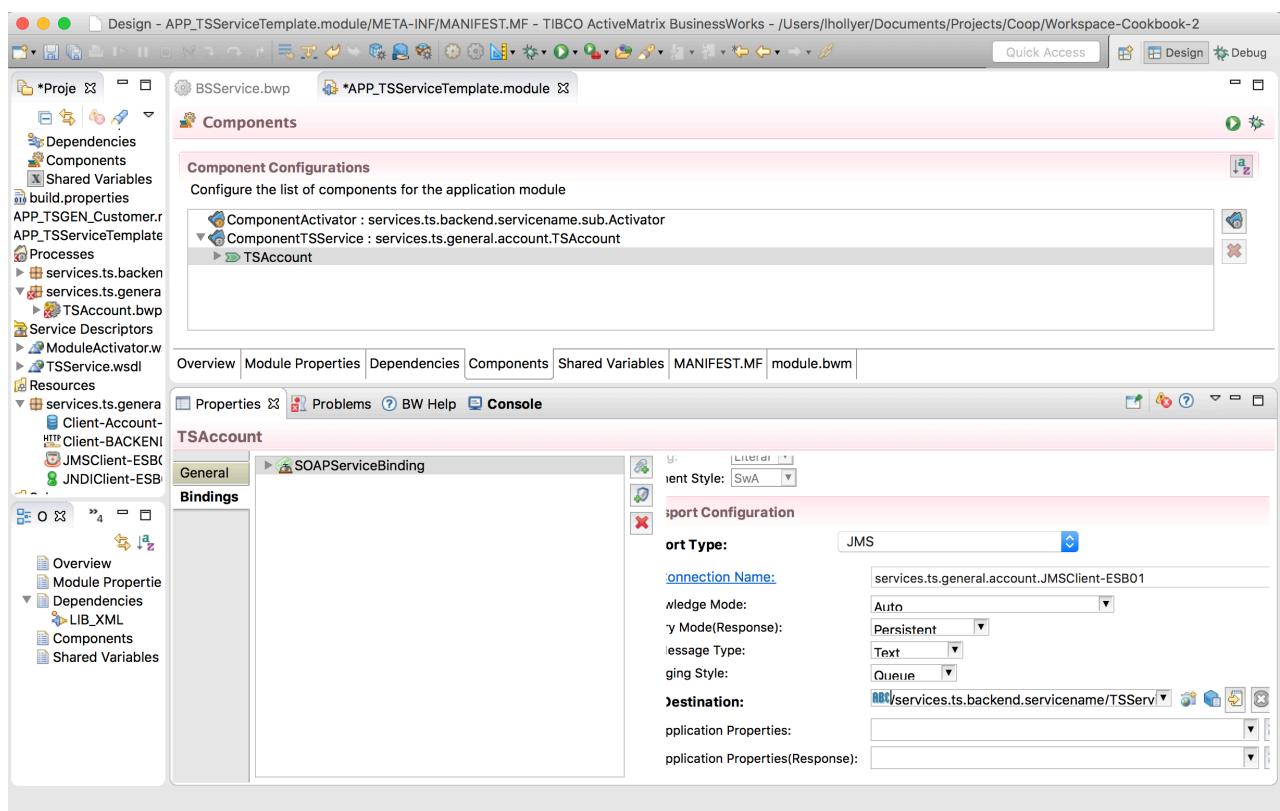
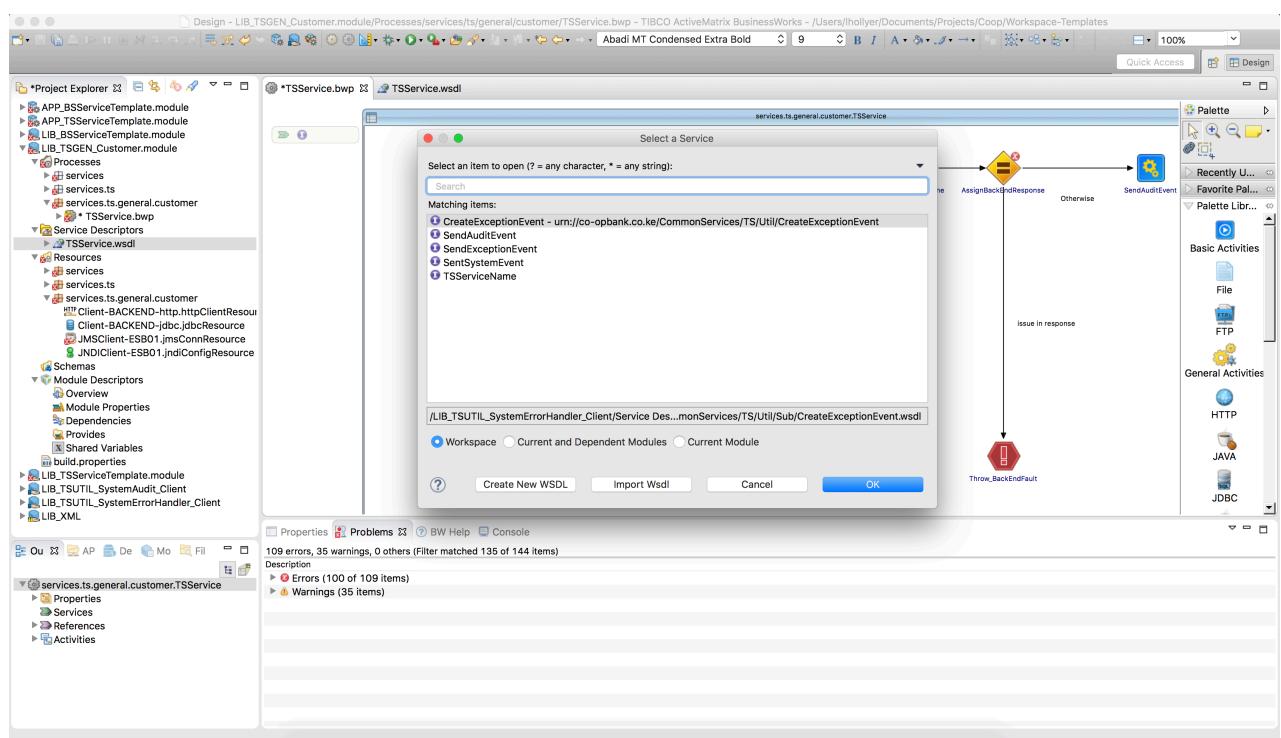


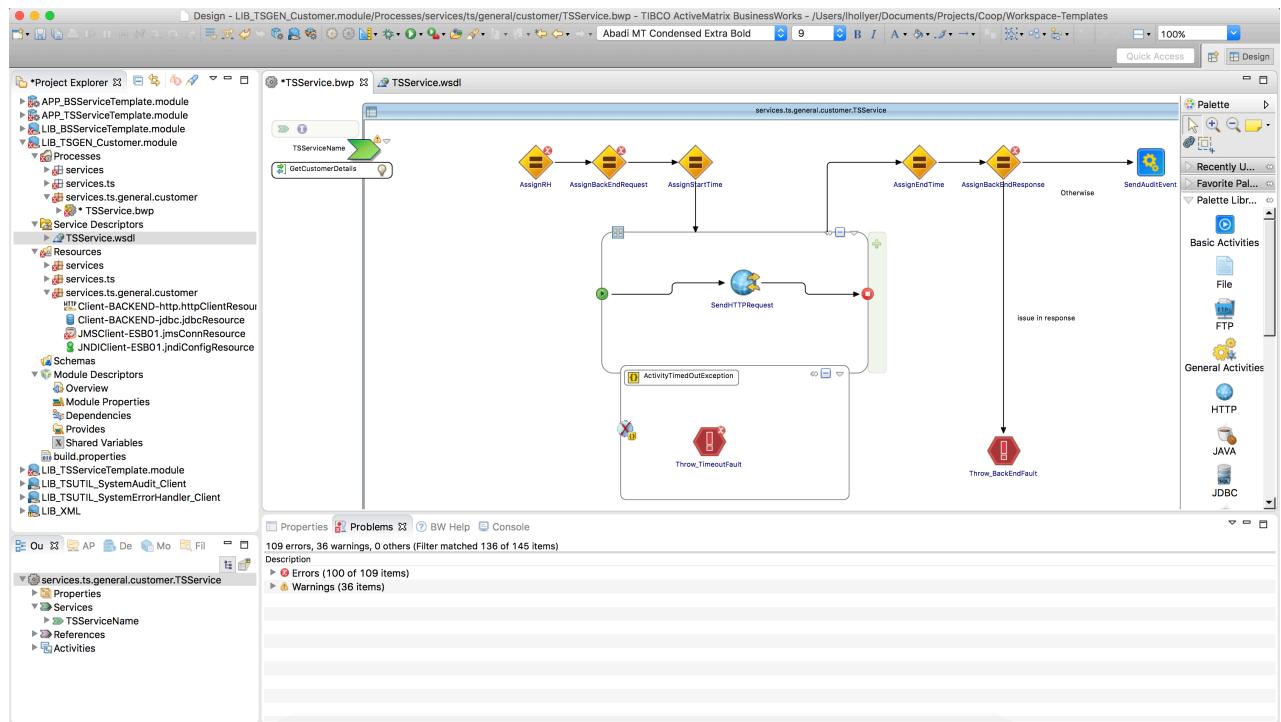
Figure 36: Create the SOAP over JMS Interface: Queue Name as Property

11. Save.

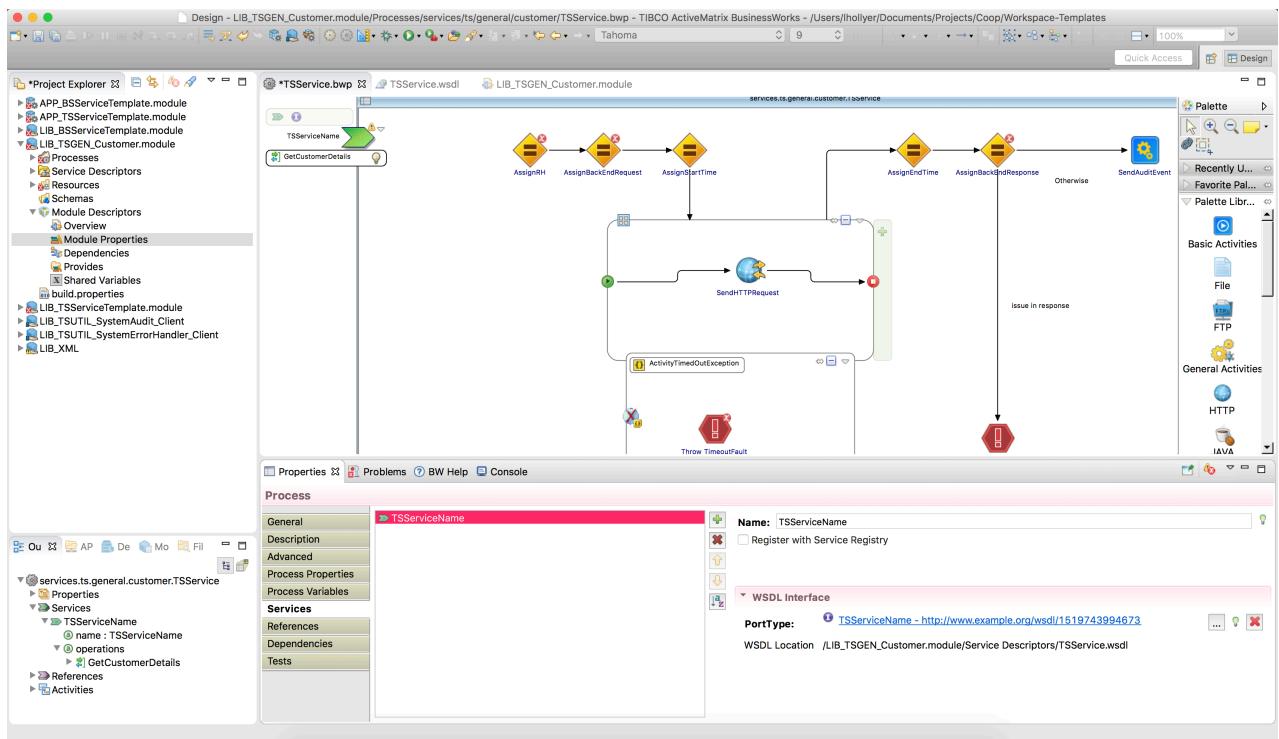
## CO-OP Bank - ESB – How To: Create BW Services from Templates



### 1. Select TS<ServiceName> and click OK.



2. Go to the process's properties and select the service on the Services tab.



### 7.3.7 Create the EMS deployment script

You must create one script that can be run against the EMS Server with tibemsadmin tool to deploy the queue required by this service.

1. In the <GIT> / trunk / EMS folder create the text file “deploy-TS<AR>\_<ServiceName>.ems” with the following content:

```
#####
# This script deploys EMS resource for the service TS<AR>_<ServiceName>
#
# Version 1.0.0 - <author> - <date>
#
# Changes:
# 1.0.0 - <date> - First version
#
#####
#
#####
# Queues

create queue <queue>
secure, sender_name_enforced, maxmsgs=100000, maxbytes=1GB, expiration=10, overflowPolicy=discardOld
grant queue <queue> user=bw send,receive

#####
commit

#####
### END OF FILE #####
#####
```

2. Replace <author>, <date>, <queue> by the appropriate values.

### 7.3.8 Create the EMS undeployment scripts

You must create one script that can be run against the EMS Server with tibemsadmin tool to undeploy the queue required by this service.

1. In the <GIT> / trunk / EMS / rollback folder create the text file “undeploy-TS<AR>\_<ServiceName>.ems” with the following content:

```
#####
# This script un-deploys EMS resource for the service TS<AR>_<ServiceName>
#
# Version 1.0.0 - <author> - <date>
#
# Changes:
# 1.0.0 - <date> - First version
#
#####
#
# Queues

delete queue <queue>

#####
#
commit

#####
### END OF FILE #####
#####
```

2. Replace <author>, <date>, <queue> by the appropriate values.

## 8 Setting up Automated Tests (SOAP over JMS)

This chapter explains how to automate tests for application modules in SoapUI using HermesJMS.

### 8.1 Pre-Requisites

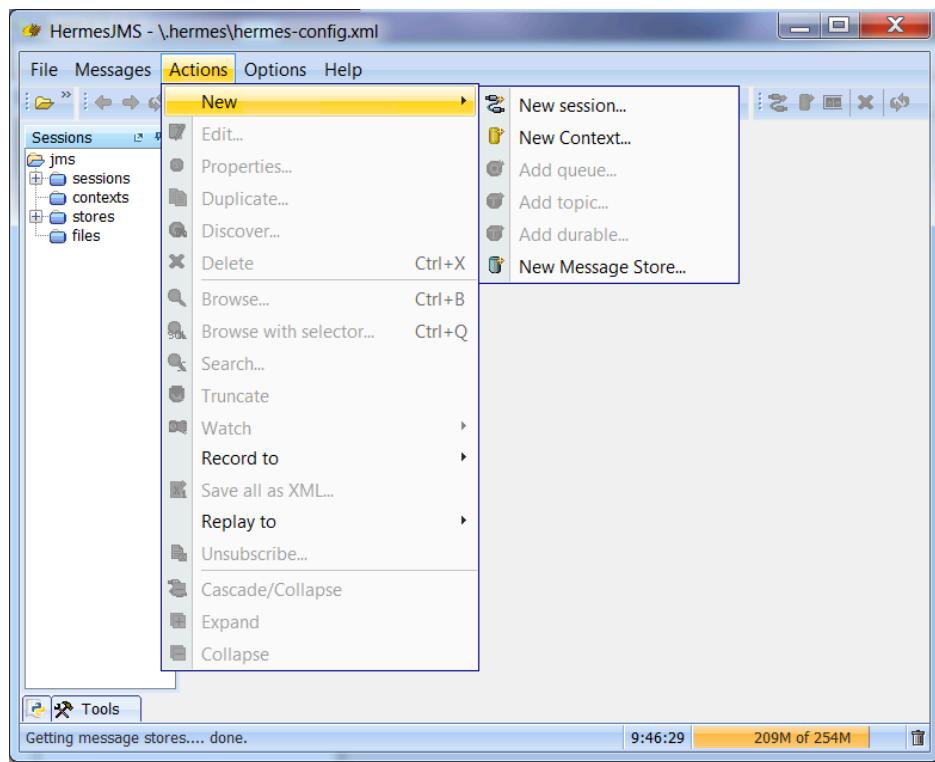
SoapUI 5.x and HermesJMS have been installed correctly. The EMS queues are up and running.

### 8.2 Set up HermesJMS

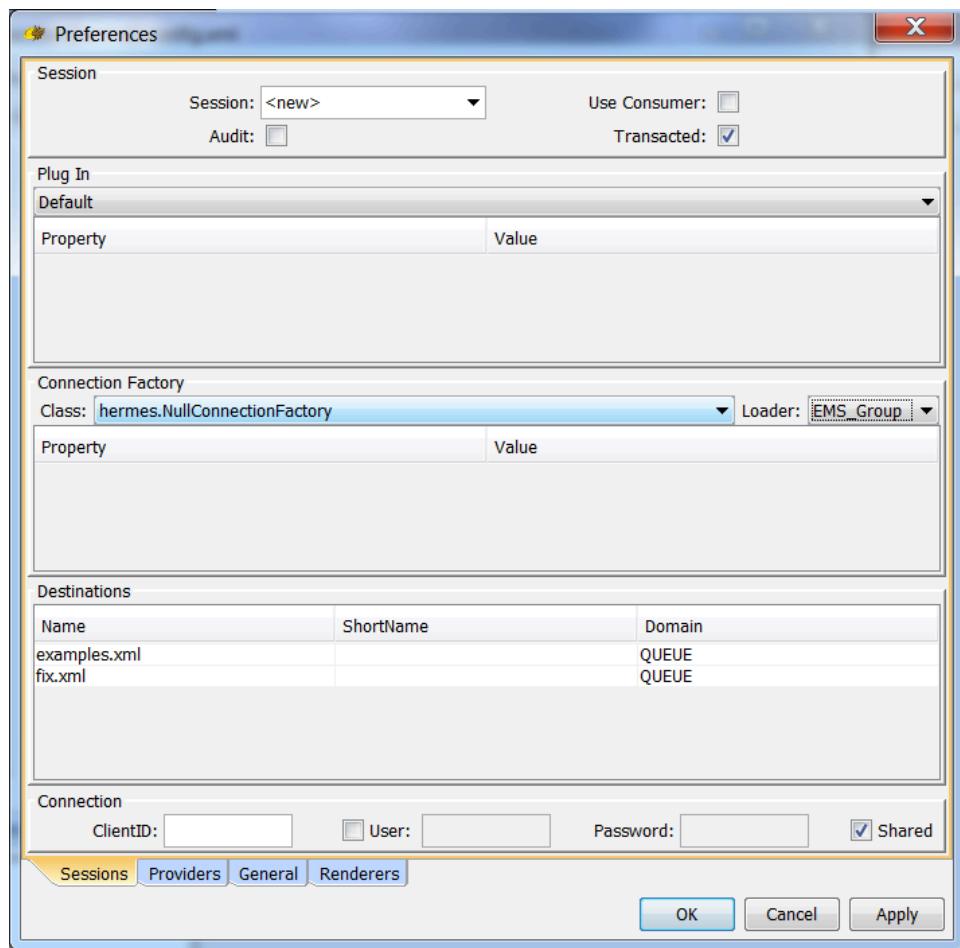
1. Open SoapUI and navigate to Tools / HermesJMS to open HermesJMS.



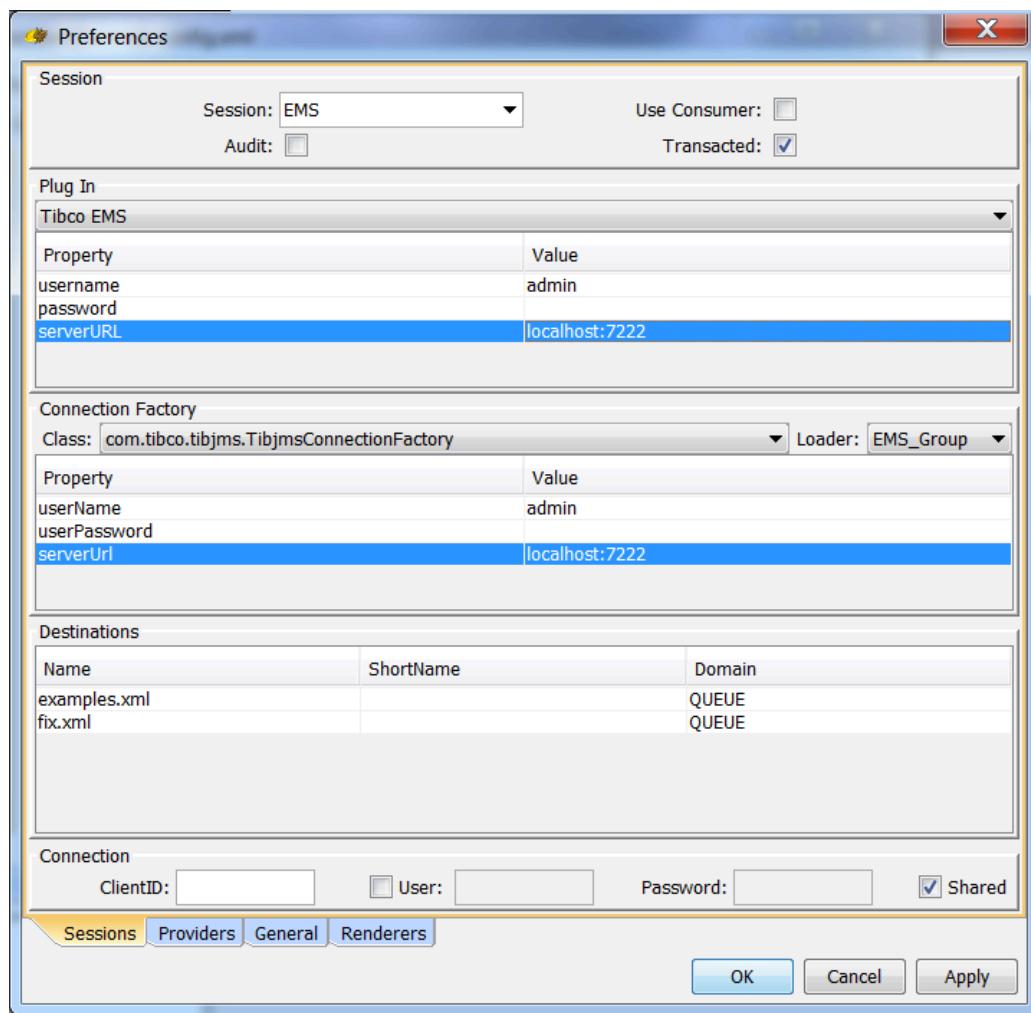
2. In HermesJMS navigate to Actions / New / New sessions to create a new session.



3. Create a new group of providers for EMS by clicking to Providers, right-clicking and adding a new group.
4. Right-click on the group and select Add JAR(s) to add the EMS JAR. Browse to <EMS\_HOME>/8.x/lib, select all JAR files and click Open and Scan.
5. Apply the changes and click OK, then right-click on the new session and click Edit.
6. Go to the Sessions tab and change the Loader under Connection Factory to the newly created EMS group.



7. Change the Connection Factory Class to **com.tibco.tibjms.TibjmsConnectionFactory** and the Plug In to **Tibco EMS**.
8. Right-click in Plug In and Connection Factory properties to add username/userName, password/userPassword, and serverURL fields. Username is **admin** and serverURL is **tcp://ec2-35-178-63-212.eu-west-2.compute.amazonaws.com:8777**



9. Click Apply and OK, then save the settings.
10. Right-click on the session and click Discover to load in the existing queues.
11. In SoapUI, create/edit the SOAP project and in Project Properties set the Hermes Config to where the .hermes folder is located (typically C:\.hermes\).
12. Change SoapUI requests endpoint to **jms://<sessionName>::queue\_<queueName>::queue\_**

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:jms="jms://aws::queue_DEMOGETDATA::queue_>
    <soap:Header/>
    <soap:Body>
        <get:GetDataRequest>
            <RequestHeader>
                <com:CorrelationID>1</com:CorrelationID>
                <mes:MessageID>1</mes:MessageID>
            </RequestHeader>
            <request>
                <AccountNumber>
                    <ns:AccountNumber>123456789</ns:AccountNumber>
                </AccountNumber>
            </request>
        </get:GetDataRequest>
    </soap:Body>
</soap:Envelope>

```

13. Under JMS Headers, check Add SoapAction and Send As Bytes Message.

JMSCorrelationID:	<input type="text"/>
JMSReplyTo:	<input type="text"/>
JMSType:	<input type="text"/>
JMSPriority:	<input type="text"/>
JMSDeliveryMode:	PERSISTENT
TimeToLive:	<input type="text"/>
Send As Bytes Message:	<input type="checkbox"/>
Add SoapAction as property:	<input checked="" type="checkbox"/>
Durable Subscription Name:	<input type="text"/>
ClientID:	<input type="text"/>
Message Selector:	<input type="text"/>

Auth Headers (0) Attachments (0) WS-A WS-RM **JMS Headers** JMS Properties (0)

14. Run the test and it should return a status code or the found record.

**N.B. SoapUI tests MUST be saved as APP\_<Area><AppName>-TEST for Jenkins to automatically run them in the provided TEST folder.**

## 9 Create POM for Maven Build

For any application a number of POM files are required to enable the Maven build to be completed by the Jenkins pipeline. There are three main POM files;

- The main parent POM
- The application POM
- The module POM

### 9.1 The Parent POM

1. The parent POM is created in the BW/POM directory using the naming convention <service name>.xml
2. In the BW/POM directory there is a template POM for both technical and business services. This appropriate file can be copied to the <service name>.xml file, for example, copy APP\_BSTemplate.xml to APP\_BSGetAccountBalance.xml.
3. In the file, edit the two sections indicated;

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.tibco.bw</groupId>

  <!-- Change service name here -->
  <artifactId>APP_BS<Service name>.module.application.parent</artifactId>

  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <modules>
    <module>../../XML/LIB_XML</module>
    <module>../TechnicalServices/LIB_TSUTIL_SystemAudit_Client</module>
    <module>../TechnicalServices/LIB_TSUTIL_SystemErrorHandler_Client</module>

  <!-- Add dependant modules in here -->
  <module>../BusinessServices/APP_BS<service name>.module</module>
  <module>../BusinessServices/APP_BS<service name>.module.application</module>
  </modules>
</project>
```

4. Ensure that the service name is correct, and the dependent modules are all correct.
5. Push this file to the GIT Repo.

### 9.2 The Application POM

1. In the top level of the Application, there needs to be a pom.xml file to enable the build of the application based on the dependencies in the parent POM.
2. Take a copy of the BW/POM/ExampleApplication.pom.xml file and copy it to pom.xml in the root of the application folder.
3. Edit the file and change the specified areas.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.tibco.bw</groupId>

    <!-- Change service name -->
    <artifactId>APP_<Service name>.module.application.parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>

    <!-- Change service name -->
    <relativePath>../../POM/APP_<Service name>.xml</relativePath>
  </parent>

  <!-- Change service name -->
  <artifactId>APP_<Service name>.module.application</artifactId>
  <packaging>bwear</packaging>
  <build>
    <plugins>
      <plugin>
        <groupId>com.tibco.plugins</groupId>
        <artifactId>bw6-maven-plugin</artifactId>
        <version>1.3.1</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
  <profiles>
    <profile>
      <id>DEFAULT</id>
      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>
      <properties>
        <profile>default.substvar</profile>
      </properties>
    </profile>
  </profiles>
</project>

```

4. Note, the reference to the parent POM is a relative path, so if the application is in a non-standard place then this path needs to be changed.
5. Push this file to the GIT repo.

### 9.3 The Module POM

1. In the top level of the module there needs to be a pom.xml file. This enables the build of the module based on the parent POM dependencies.
2. Take a copy of the BW/POM/ExampleModule.pom.xml file into the top-level root directory of the module called pom.xml. This pom.xml file can then be modified.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.tibco.bw</groupId>

  <!-- Change service name -->
  <artifactId>APP_<service name>.module.application.parent</artifactId>
  <version>1.0.0-SNAPSHOT</version>

  <!-- Change service name -->
  <relativePath>../../POM/APP_<service name>.xml</relativePath>
  </parent>

  <!-- Change service name -->
  <artifactId>APP_<service name>.module</artifactId>
  <packaging>bwmodule</packaging>
  <dependencies>
    <dependency>
      <groupId>com.tibco.plugins</groupId>
      <artifactId>com.tibco.bw.palette.shared</artifactId>
      <version>6.1.100</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <outputDirectory>target/classes</outputDirectory>
    <plugins>
      <plugin>
        <groupId>com.tibco.plugins</groupId>
        <artifactId>bw6-maven-plugin</artifactId>
        <version>1.3.1</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
</project>

```

3. Note, the reference to the parent POM is a relative path, so if the application is in a non-standard place then this path needs to be changed.
4. Push this file to the GIT repo.

## 10 Create Service Profile for Jenkins

In order for Jenkins to deploy and run the newly made application, the service profile needs to be created. This is done by running **Jenkins / createAppconfig.bat** which automatically creates the profile XML.

1. In command prompt, run the command **createAppconfig.bat <application name>**

The service config XML has been created under **Jenkins / ServiceProfiles / <appname>.xml**

2. Commit the code to Git.