Oracle SOA Suite 11g R1 PS5

# SOA Suite for healthcare integration Series

## Exception Handling - Processing Endpoint Errors

michael@czapski.id.au

January 2013

## Table of Contents

Introduction ..... 1

Solution Overview ..... 2

Modify Inbound Endpoint to use Functional ACKs and Validate ..... 3

Exception Handler Application ..... 5

Start the "Receiving" "External Systems" ..... 15

Send valid ADT messages ..... 16

Send invalid ADT messages ..... 17

Summary ..... 20

## Introduction

In this article we will add exception handling to the routing and transformation solution developed in the article entitled "SOA Suite for healthcare integration Series – Domain Value Map (DVM) – On-the-fly Code Mapping", to explore how message processing exceptions can be redirected to a logic component for handling. The exception handling components will handle exceptions form all SOA Suite for healthcare integration applications.

This article assumes that the reader has the SOA Suite for healthcare integration environment with all necessary components installed and ready to use. The Bill of Materials for such an environment and a discussion on where the components can be obtained is provided in the earlier article, "SOA Suite for healthcare integration Series - Overview of the Development Environment", to be found at http://blogs.czapski.id.au/2012/08/soa-suite-for-healthcare-integration-series-overview-of-the-development-environment.

This article assumes that the reader completed the solution discussed in the earlier article, "SOA Suite for healthcare integration Series – Domain Value Map (DVM) – On-the-fly Code Mapping", to be found at http://blogs.czapski.id.au/2013/01/soa-suite-for-healthcare-integration-series-domain-value-map-dvm-on-the-fly-code-mapping.

## Solution Overview

In healthcare messaging environments it is expected that exceptions in message processing will be handled in a reasonable way, not by simply dropping messages without some form of notification being provided to the sender or to the operations staff.

The SOA Suite for healthcare integration infrastructure will queue invalid incoming messages, or outgoing messages which cannot be delivered, to the JMS B2B_IN_QUEUE, if JMS is configured as the default transport (which it is for this article series). Messages in error are queued with a JMS User Defined Property "MSG_TYPE" set to the value of "3". An inbound message can cause an exception in a number of circumstances, some of which are:

- message cannot be decoded (for example not a HL7 v2 delimited)
- message structure for the message cannot be identified (for example ORU coming in when no ORU document is defined)
- Endpoint cannot be identified
- …

An outbound message can cause an exception in a number of circumstances, some of which are:

- Endpoint cannot be identified so the message cannot be sent
- External target systems is down and no retry is configured
- …

The solution components are depicted in Figure 1 Solution Components.

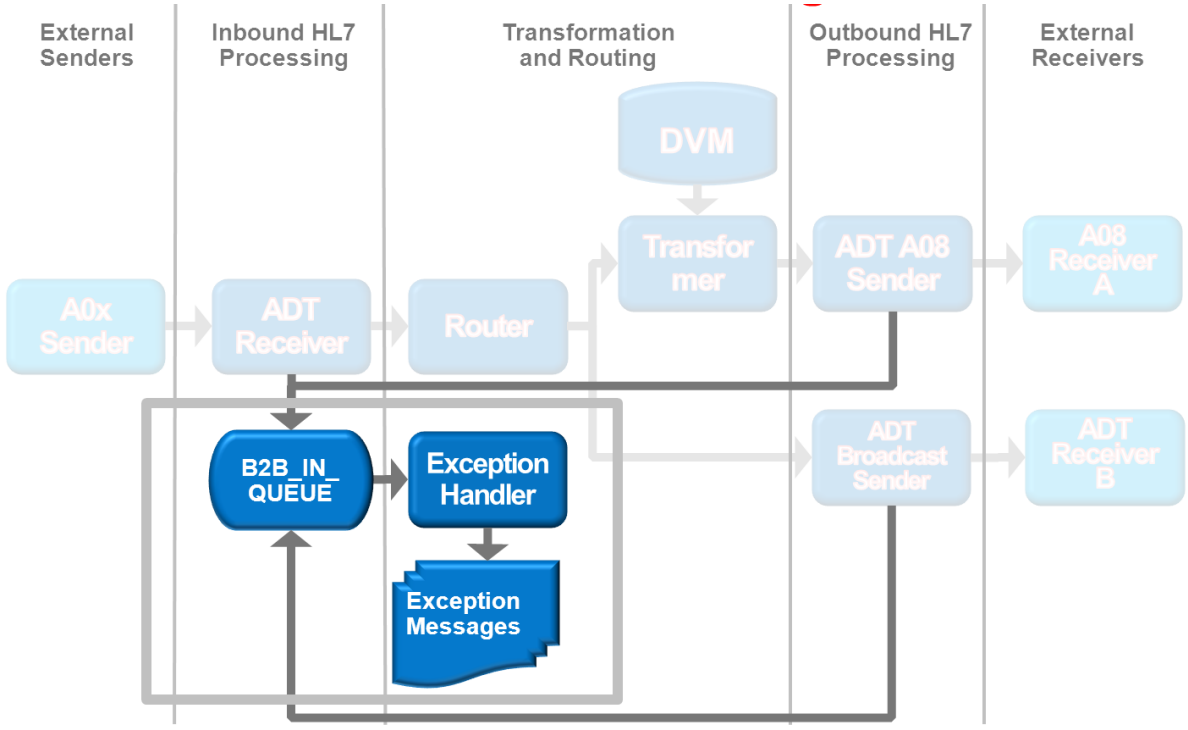



Figure 1 Solution Components

The diagram uses the convention which clearly separates the external systems, the SOA Suite for healthcare integration-specific components and generic SOA Suite components using the "swim-line" analogy.

The solid lines from the "ADT Receiver", "ADT A08 Sender" and "ADT Broadcast Sender" to the "B2B_IN_QUEUE" JMS Queue depict implicit integration between the endpoints and the JMS infrastructure used to both pass messages between the endpoints and the SOA Composites and to queue exception messages to be processed as might be desired. This latter functionality is what this article will explore.

Exception handling functionality revolves around the SOA Suite for healthcare integration infrastructure passing to a SOA Composite a message and a property whose value allows the SOA Composite to de-queue the message from the queue and process it as might be required.

This topic is discussed in some detail in the Oracle "Technical Note #007, Exception Handling", available at http://www.oracle.com/technetwork/testcontent/b2b-tn-007-exception-handling-133087.pdf. While it applies to the Oracle B2B infrastructure it is equally applicable to the "SOA Suite for healthcare integration". Importantly, the tech note provides the definition of the XML structure which is used to represent exception messages in the SOA Suite for healthcare integration.

To trigger exception processing we will submit an invalid HL7 v2 message to the existing solution, will receive a HL7 v2 message, validate it, acknowledge it with functional ACK and pass it onto the SOA Composite if it is valid. If the message is not valid an exception message will be delivered to the exception handler, which we will develop in this article, and will be subsequently written to a file in the file system. Clearly, a more imaginative / useful / elaborate handling of exception messages is possible. Developing such a sophisticated solution is a SOA Suite project and has nothing to do with HL7 processing as such.

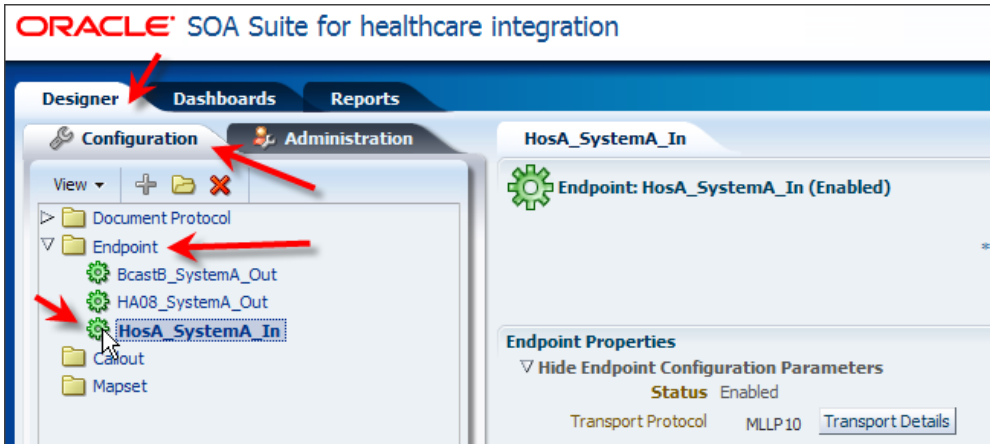
We need to create a SOA Composite which will use JMS selectors to read messages from the pre-configured JMS Queue and will write them to a file in the file system.

## Modify Inbound Endpoint to use Functional ACKs and Validate

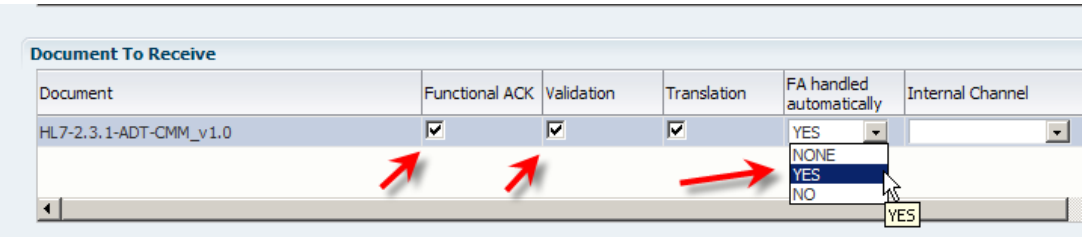
As configured, the HosA_SystemA_In endpoint, which receives ADT messages, will send a canned ACK as soon as it receives a message. It is also configured to not validate messages it receives so the endpoint will not trigger exception processing if the message is invalid. Even with this there will be circumstances when exceptions will occur and trigger exception handling. Some of these will be inability to deliver messages to the external systems by the outbound endpoint, for example. To make it easy for us to induce exception handling we will re-configure the inbound endpoint, HosA_SystemA_In, to use Functional Acknowledgements and to Validate messages. In this section we will go through the steps necessary to accomplish this.

- ☐ Start the "Healthcare Integration Console", http://localhost:7001/healthcare, and log in with your credentials (perhaps weblogic/welcome1)

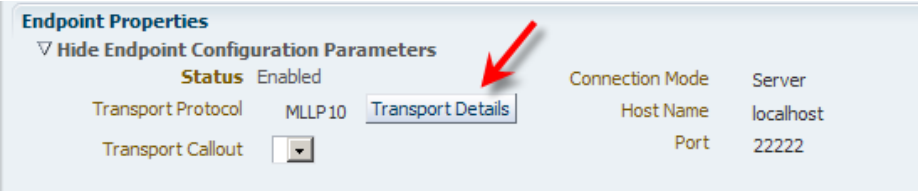
- ☐ Navigate through "Designer" → "Configuration" → "Endpoint" and double-click the "HosA_SystemA_In" endpoint name to open its configuration properties



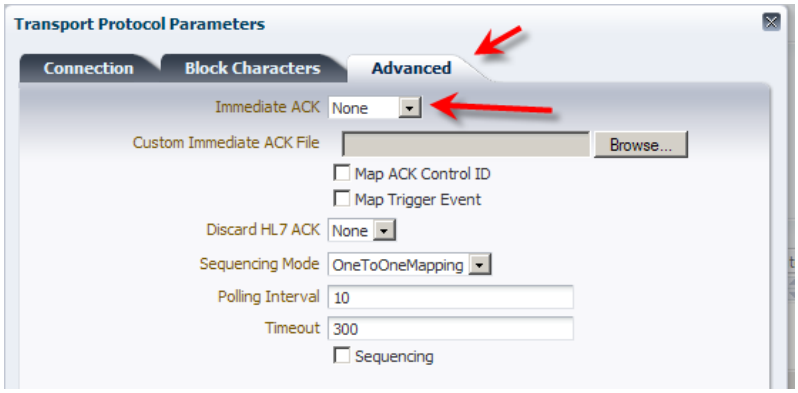

☐ In the "Document to Receive" check the "Functional ACK" and "Validate" checkboxes and choose "YES" from the "FA handled automatically" drop down




☐ Click the "Transport Details" button




☐ Click the "Advanced" Tab, change "Immediate ACK" to "none" and click "OK"




☐ Click "Apply" to complete the procedure

The endpoint is re-configured as required and changes are applied to the runtime. Now we will develop the composite application which will pick up exception messages form the JMS Queue and write them to a file in the file system.

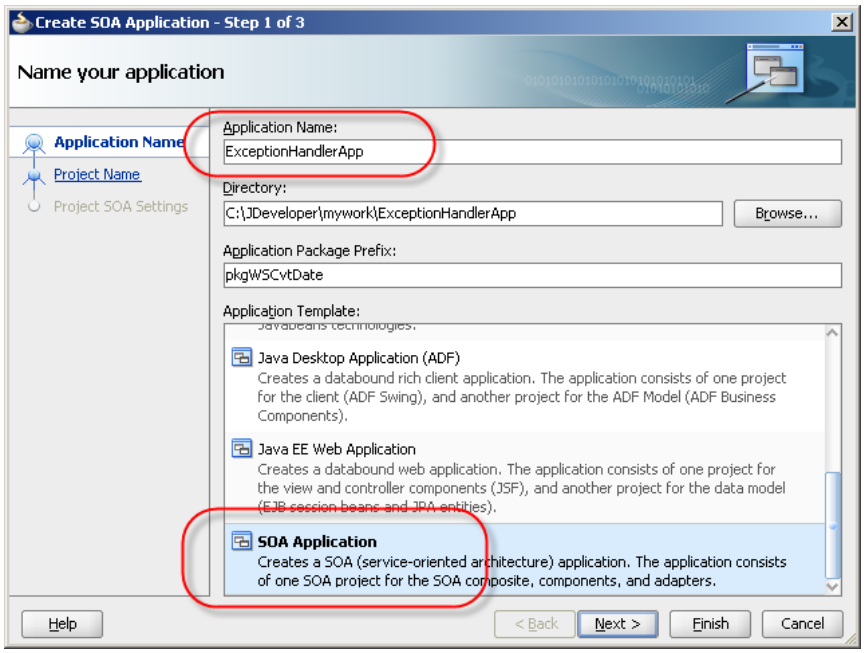
## Exception Handler Application

In this section we develop and deploy the exception handler application, "ExceptionHandlerApp", to receive selected JMS messages and write them to files in the file system.

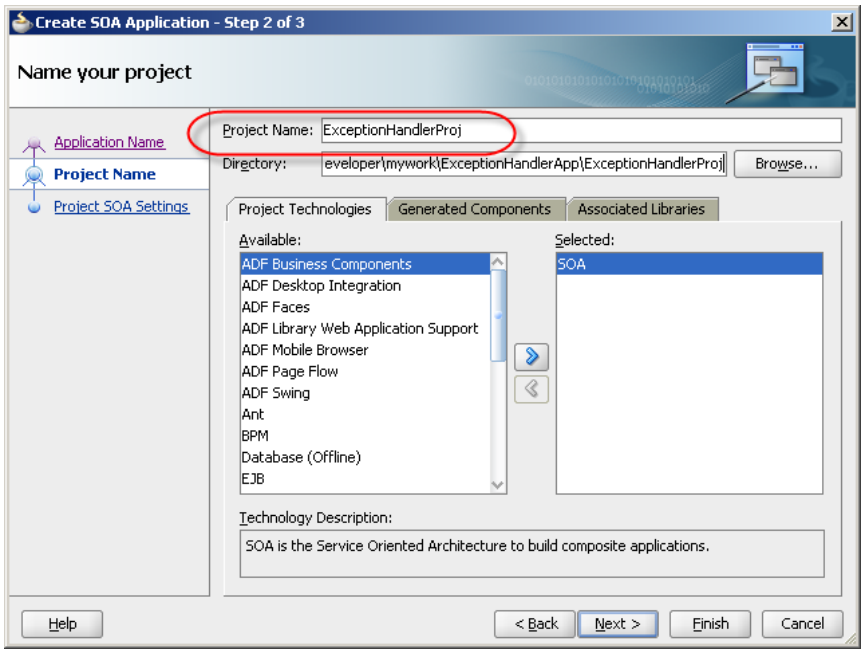
☐ Start JDeveloper Studio, if it is not already running

☐ Pull down the "Application" menu and choose "New…"

☐ Name the new application "ExceptionHandlerApp", choose "SOA Applicaiotn" and click "Next>"



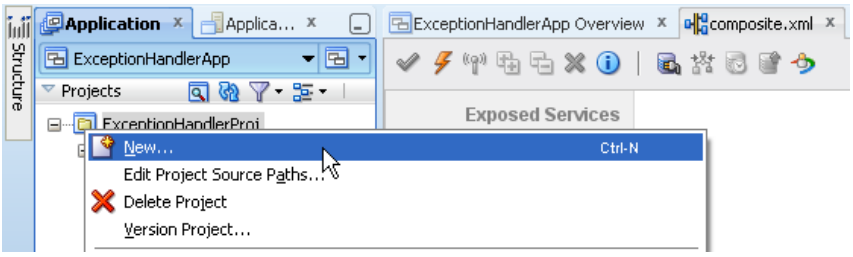

☐ Name the new project "ExceptionHandlerProj" and click "Finish"



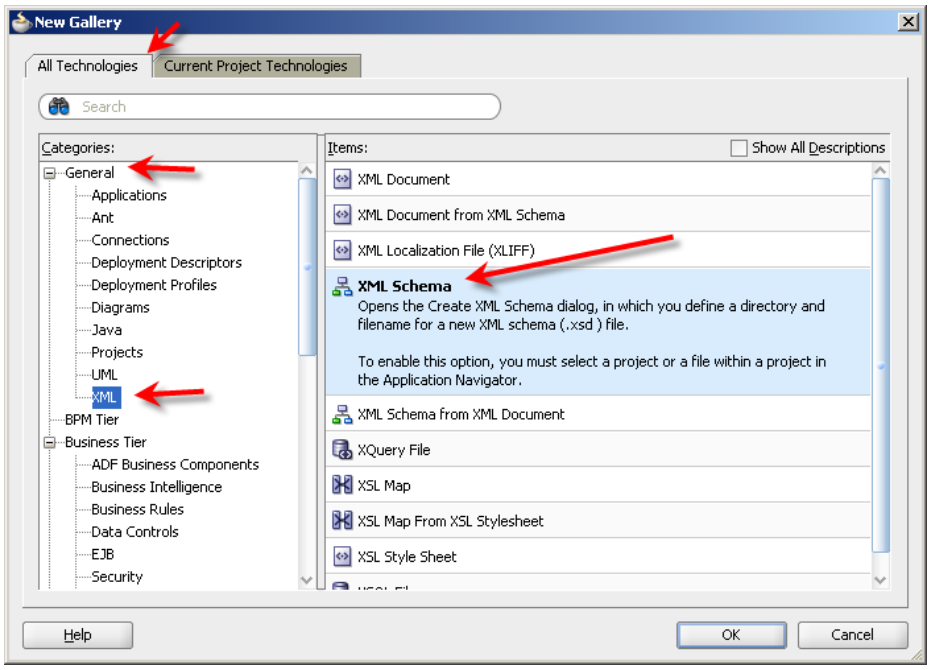
As mentioned in the introduction, the Oracle "Technical Note #007, Exception Handling", available at http://www.oracle.com/technetwork/testcontent/b2b-tn-007-exception-handling-133087.pdf, provides the definition of the XML structure which is used to represent exception messages in the SOA Suite for healthcare integration.

We need to create an XML Schema Definition (XSD) document in our project to use as the definition of the exception message.

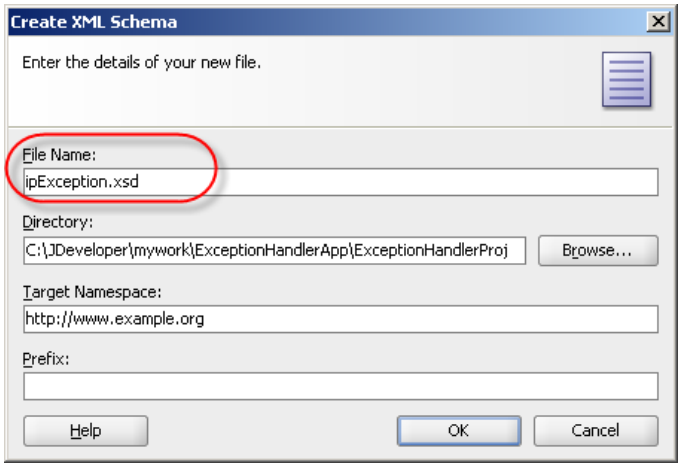
☐ Right-click the "ExceptionHandlerProj" project name and choose "New…"



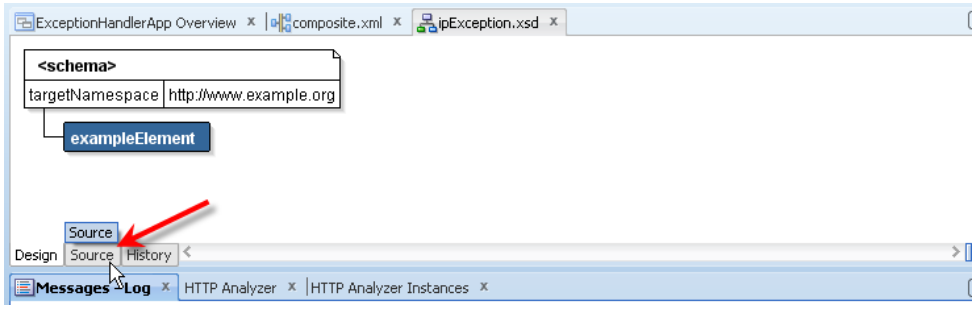

☐ Click the "All Technologies" Tab, expand the "General" Tab, click "XML" to show the list of XML-related technologies, choose "XML Schema" and click "OK"




☐ Name the new XML Schema "ipException.xsd" then click "OK"




☐ Click the "Source" Tab to switch to the source code mode

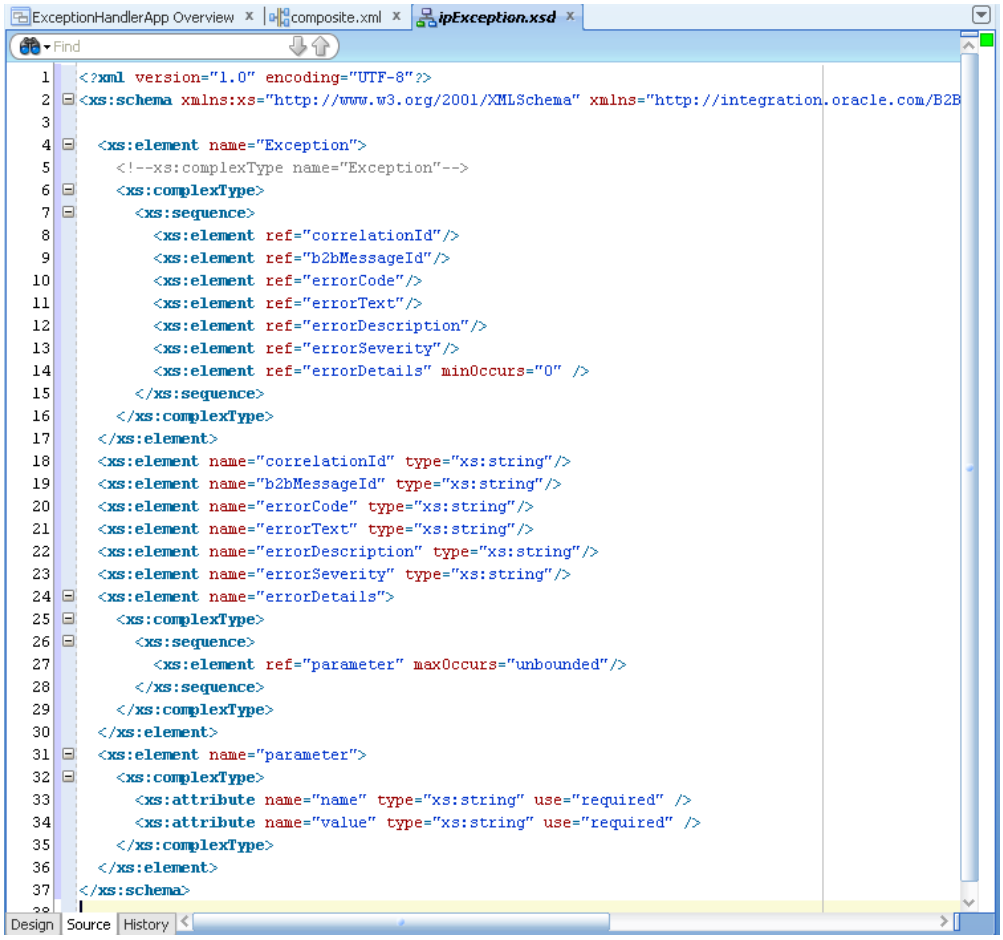



☐ Open a web browser, navigate to the Technical Note #007, http://www.oracle.com/technetwork/testcontent/b2b-tn-007-exception-handling-133087.pdf, scroll down to the bottom of the document and copy the XML Schema definition – reproduced below for convenience

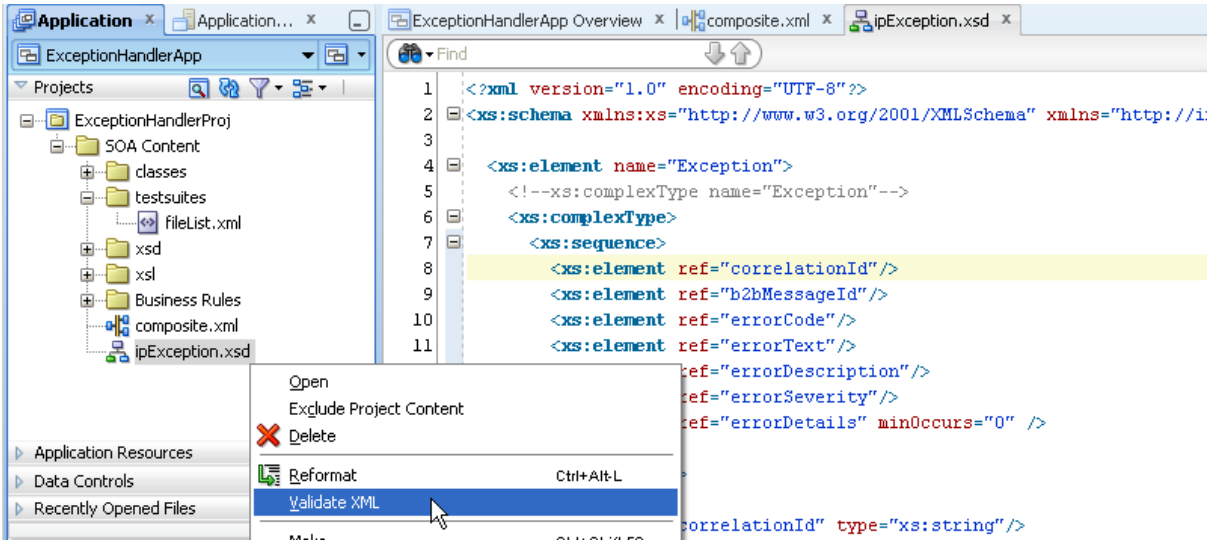
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://integration.oracle.com/B2B/Exception"
targetNamespace="http://integration.oracle.com/B2B/Exception">

  <xs:element name="Exception">
    <!--xs:complexType name="Exception"-->
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="correlationId"/>
        <xs:element ref="b2bMessageId"/>
        <xs:element ref="errorCode"/>
        <xs:element ref="errorText"/>
        <xs:element ref="errorDescription"/>
        <xs:element ref="errorSeverity"/>
        <xs:element ref="errorDetails" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="correlationId" type="xs:string"/>
  <xs:element name="b2bMessageId" type="xs:string"/>
  <xs:element name="errorCode" type="xs:string"/>
  <xs:element name="errorText" type="xs:string"/>
  <xs:element name="errorDescription" type="xs:string"/>
  <xs:element name="errorSeverity" type="xs:string"/>
  <xs:element name="errorDetails">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="parameter" maxOccurs="unbounded"/>
      </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
  <xs:element name="parameter">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="value" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

☐ Paste the XSD text into the new XSD definition in JDeveloper, replacing the default content

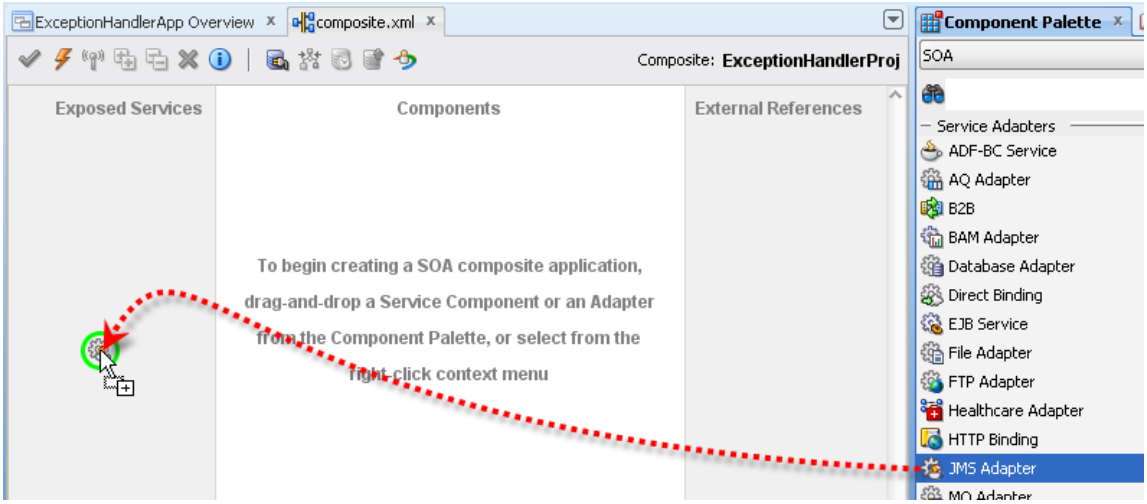



☐ In the project node tree right-click the "ipException.xsd" node and choose "Validate XML" to make sure that the schema XML is valid. Correct any errors you may find

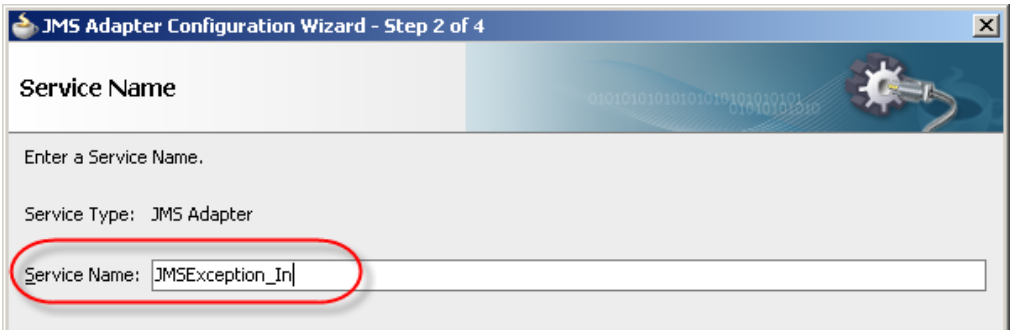



☐ Click the "Save All" button to save the changes to the project so far and close the XSD editor to return to the composite editor

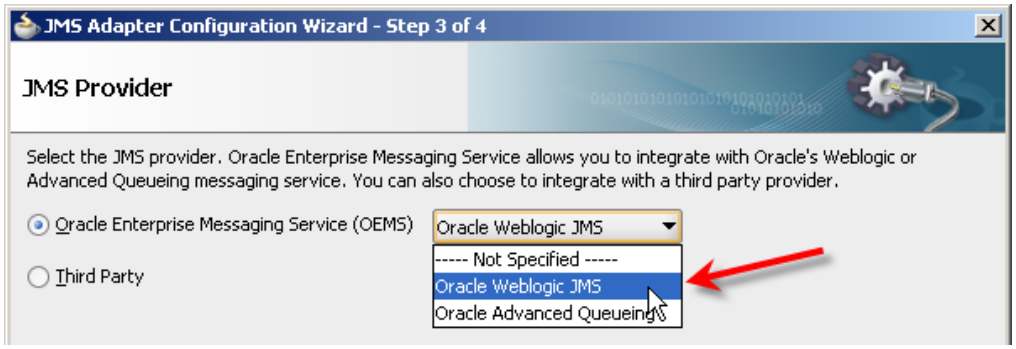
☐ From the "Component Palette" → "SOA" → "Service Adapters" drag the "JMS Adapter" to the "Exposed Services" swim line to start the JMS configurator wizard



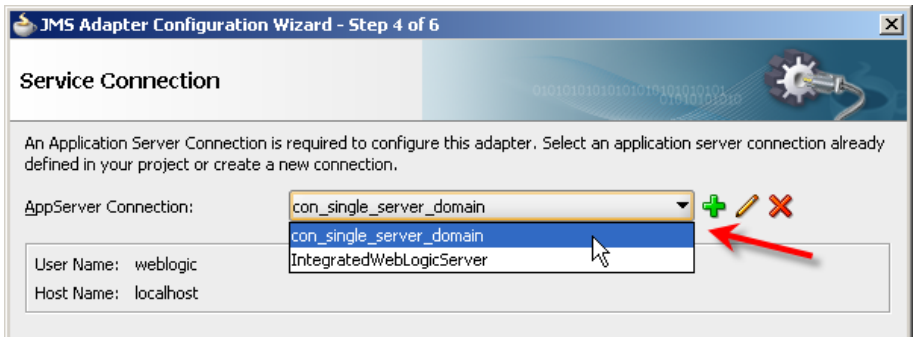

☐ Name the service "JMSException_In" and click "Next>"



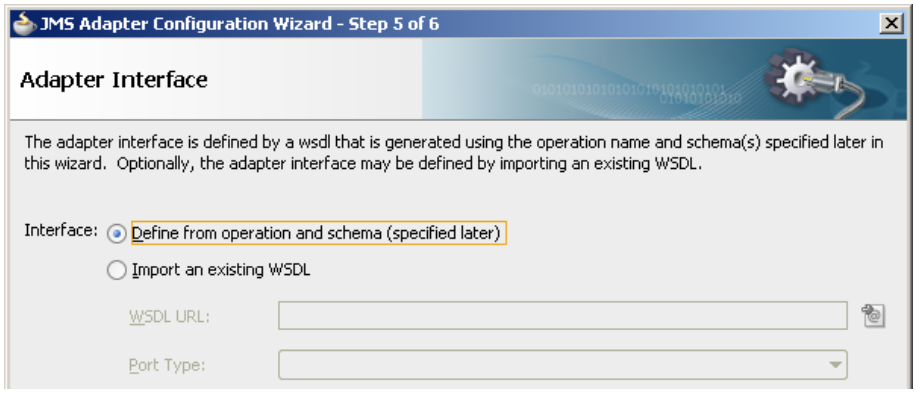

☐ Choose "Oracle Weblogic JMS" as the OEMS and click "Next>"



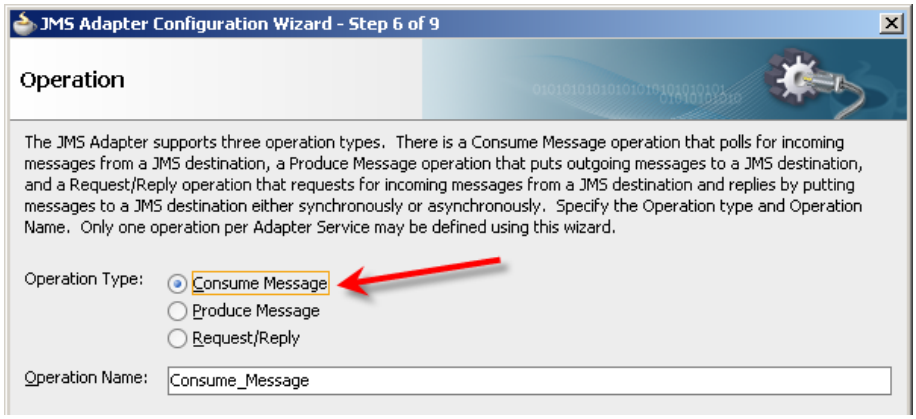

☐ Choose your "AppServer Connection" and click "Next>"



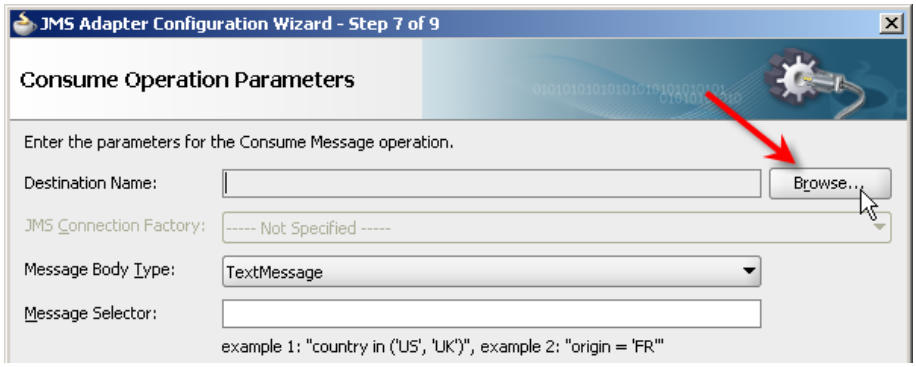

☐ Accept default for "Adapter Interface" and click "Next>"



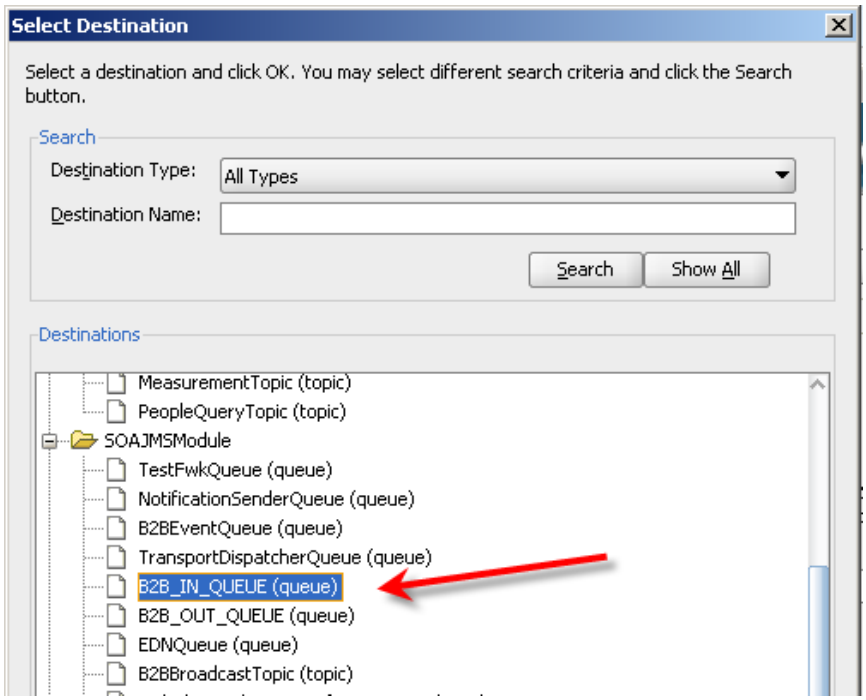

☐ Choose "Consume Message" operaton and click "Next>"



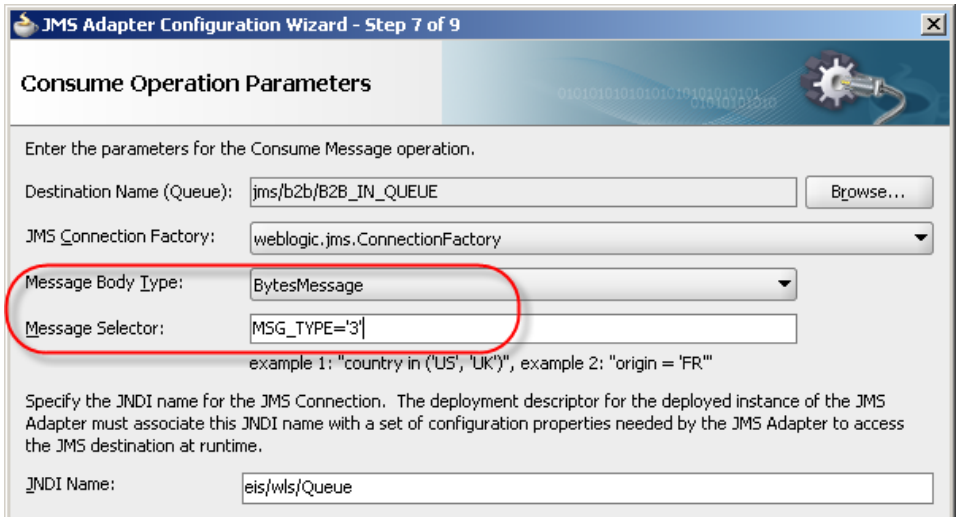

☐ Click the "Browse…" button alongside the "Destination Name"



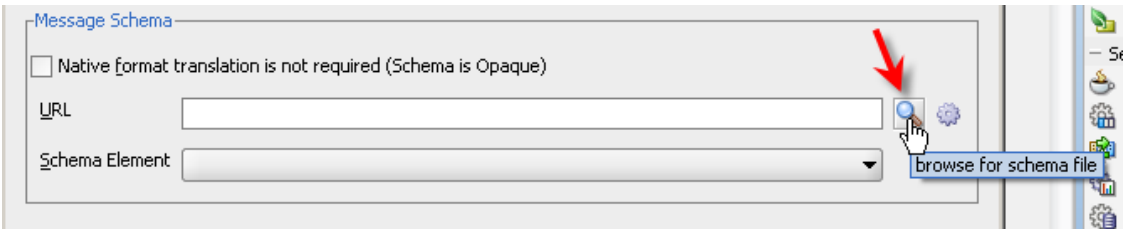

☐ Scroll down and choose "B2B_IN_QUEUE" and click "OK"



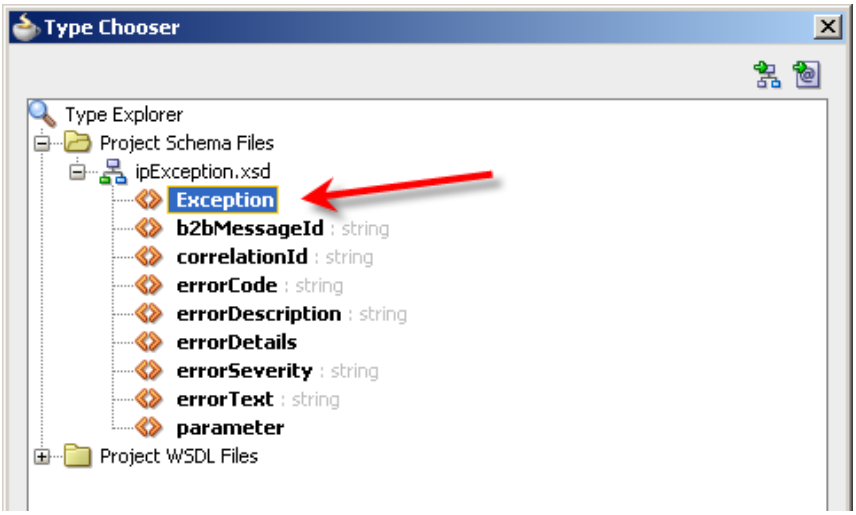

☐ Change the "Message Body Type" to "BytesMessage", enter "MSG_TYPE='3'" into the Message Selector box and click "Next>"



☐ Click the "Browse for schema file" button



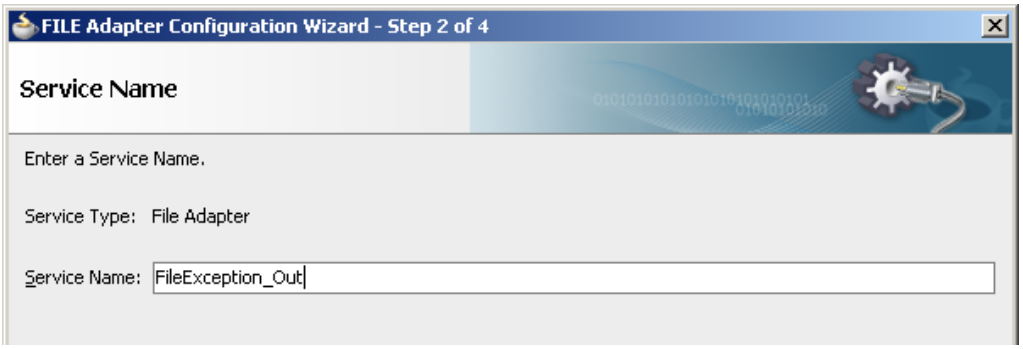

☐ Expand the node tree, choose "Exception" and click "OK"



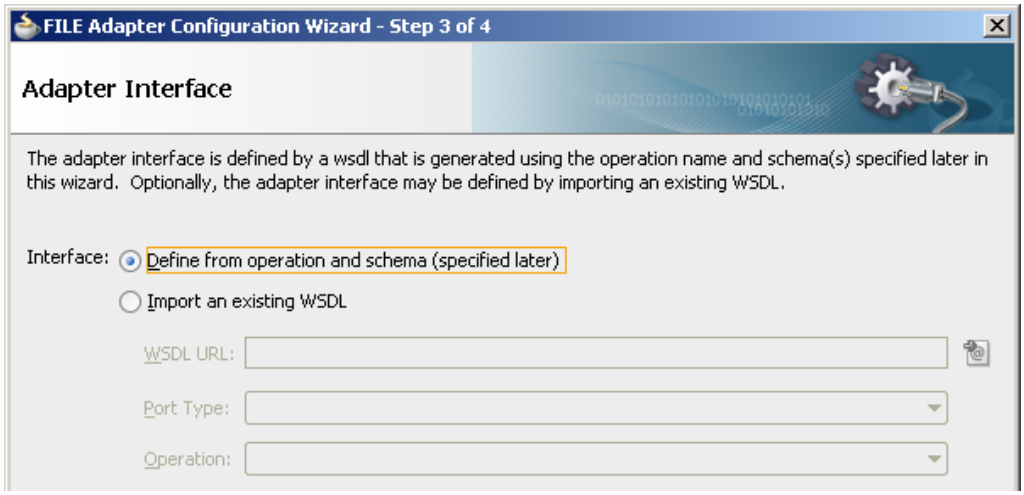

☐ Click "Next" and "Finish" to complete JMS Adapter confguratoin

☐ From the "Component Palette" → "SOA" → "Service Adapters" drag the "File Adapter" to the "Exposed Services" swim line to start the JMS configurator wizard

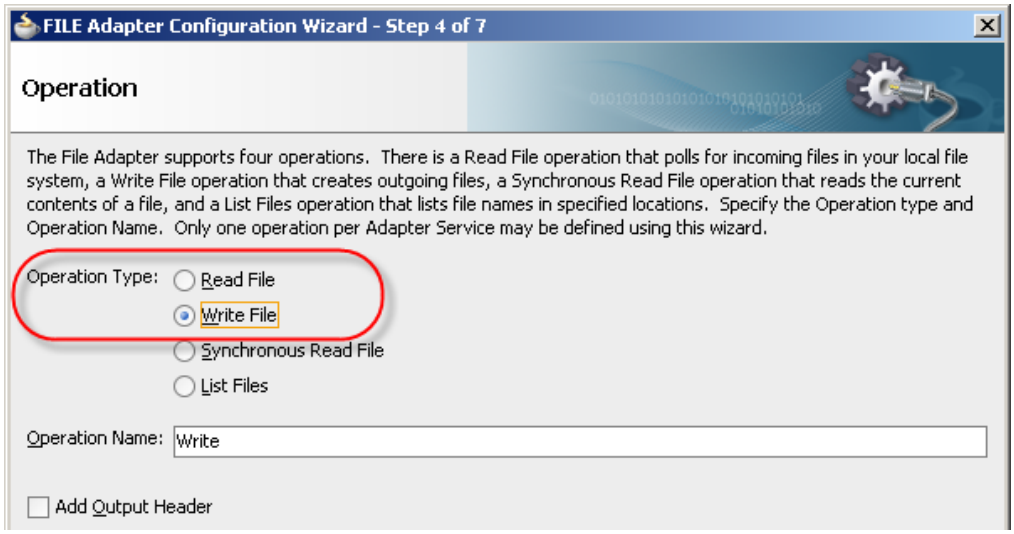
☐ Enter "FileException_Out" as the service name and click "Next>"



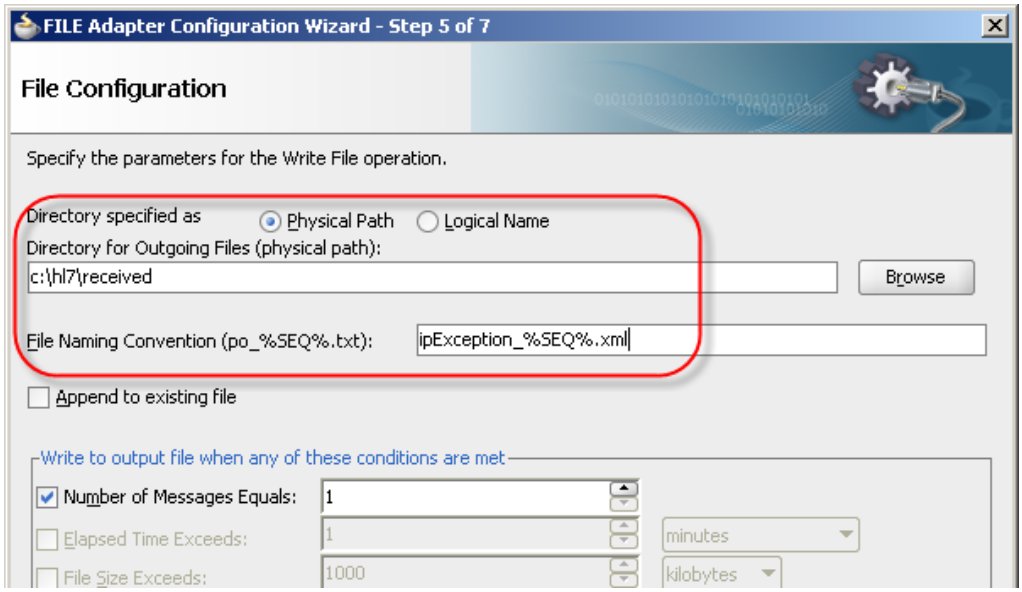

☐ Accept the default for the "Adapter Interface" and click "Next>"



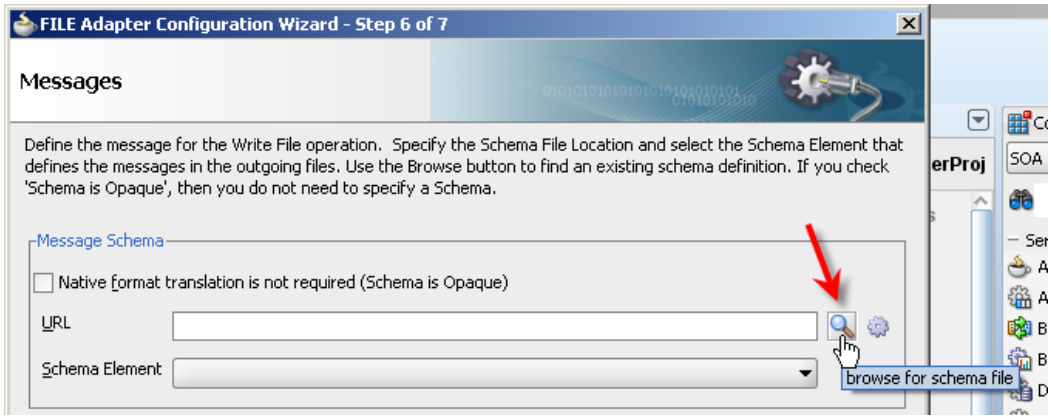

☐ Choose the "Write File" operation and click "Next>"




☐ Enter the file system path to a directory where the exception files will be written, enter the file name, "ipException_$SQ%.xml" and click "Next>" – files will have a name followed by an automatically-generated sequence number and the extension of ".xml"



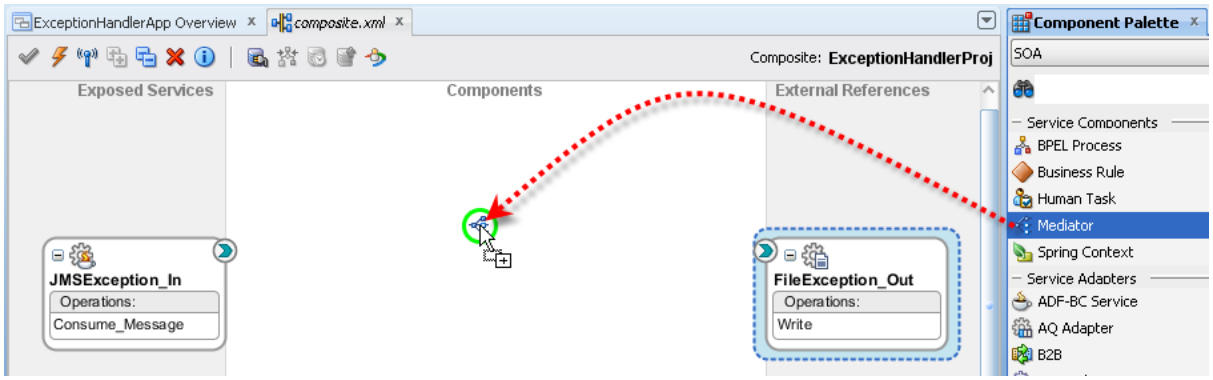

☐ Click the "Browse for schema file" button to locate the xml schema



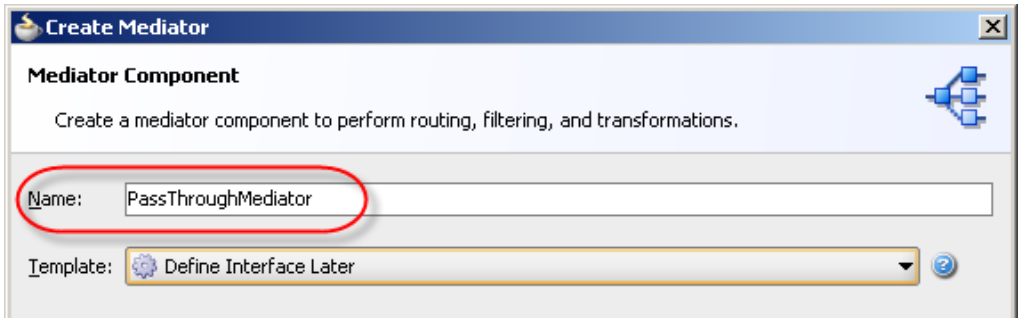

☐ Expand the "ipException.xsd", choose the "Exception" element and click "OK"


☐ Click "OK" then "Next>" and "Finish" to complete the wizard

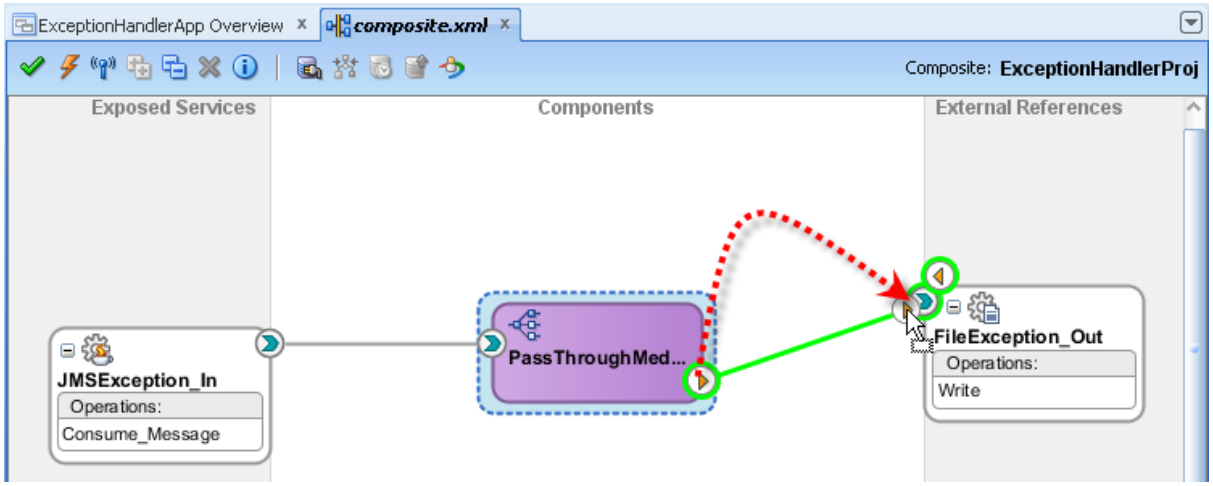
☐ From the "Service Components" list drag the "Mediator" component to the "Components" swim line




☐ Name the new component "PassThroughMediator" and click "OK"




☐ Connect adapters to the component




☐ Click the "Save All" button, right-click on the name of the project and deploy the project

Note that if you had exceptions in any of the work done so far, all accumulated exception messages will be written to files in the file system. Inspect some of them to see what they are pertaining to then delete them before proceeding.

## Start the "Receiving" "External Systems"

We will use the CMDHL7Listener command line client to receive HL7 ADT messages look at the output in the output directory specified on the listener's command line – for me c:\hl7\received. The CMDHL7Listener will display trace of message exchange in the console

window. The SOA Suite for healthcare integration will record message tracking information which we will look at a later stage.

Please note that in this solution the CMDHL7Listener returns an ACK as soon as it gets the message.

☐ Check that your configured output directory is empty

☐ To start the listener for the "BcastB_SystemA_Out" sender endpoint, in a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Listener_v0.7.jar -c ID_ -p 22200 -s
c:\hl7\received
```

☐ Inspect the CMDHL7Listener console output making sure the listener started and is listening on the appropriate port

```
08/12/2012 10:45:36 AM au.id.czapski.hl7.CMDHL7Listener main
INFO: Port: 22200
08/12/2012 10:45:36 AM au.id.czapski.hl7.CMDHL7Listener main
INFO: Store in: c:\hl7\received
08/12/2012 10:45:36 AM ca.uhn.log.HapiLogImpl info
INFO: au.id.czapski.hl7.SimpleACKApplication registered to handle *^*
messages
08/12/2012 10:45:36 AM ca.uhn.log.HapiLogImpl info
INFO: SimpleServer running on port 22200
```

☐ To start the listener for the "HA08_SystemA_Out" sender endpoint, in a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Listener_v0.7.jar -c ID_ -p 22100 -s
c:\hl7\received
```

## Send valid ADT messages

As before, we will use the CMDHL7Sender command line client to read files containing a single HL7 ADT message and submit them to the ADT Receiver endpoint. We will then look at the console output produced by the CMDHL7Listener which we started earlier, then look at the output in our configured output directory – for me c:\hl7\received, and finally review message tracking information in the Healthcare Integration Console.

Please note that in this solution the receiver endpoint returns immediate ACK as soon as it gets the message. There may be a delay, most noticeable the first time one executes the processing flow after application server restart, between the receipt of the ACK and the time the message is written to a file in the file system.

☐ Check that your configured output directory is empty

☐ Locate the input file containing a single HL7 message - for me this will be C:\hl7\adt\sources\ADT_A01_output_1.hl7

The content of my file, where each segment starting with the 3 character segment ID in bold text is a single line up to the next 3 character segment ID, looks like this:

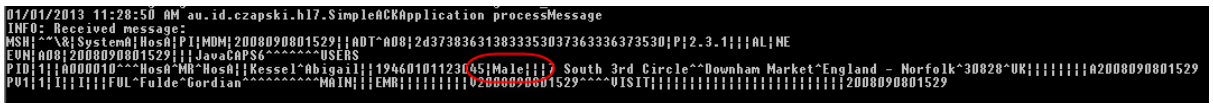
```
MSH|^~\&|SystemA|HosA|PI|MDM|2008090801529||ADT^A01|000000_CTLID_2008
090801529|P|2.3.1|||AL|NE
EVN|A01|2008090801529|||JavaCAPS6^^^^^^^USERS
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7
South 3rd Circle^^Downham Market^England -
Norfolk^30828^UK||||||||A2008090801529
PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^^MAIN|||EMR|||||||||V200809080
1529^^^^VISIT|||||||||||||||||||||||||2008090801529
```

☐ In a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Sender_v0.7.jar -a SystemA -b HosA -c ID_
-n 1 -d \r\r\n -p 22222 -h localhost -t 30000 -f
c:\hl7\adt\sources\ADT_A01_output_1.hl7
```

☐ Locate the output files in the received directory and inspect them to confirm that a) A01 or A03 and A08 have been written and b) that A01/A03 has the same content as the input file and the A08 has the same content as the input file except for the tree changes we made in the "TransformerMediator" component. Take particular note of the Gender value – which will be "Male"

The content of the A08 output file looks like this:




The content of the file named "383133323336303138313537343833532_ADT_A08_1356512791863.hl7" is the same as the message which was sent except for the fields and components which we explicitly modified.

☐ Submit the ADT A03 file, `ADT_A03_output_1.hl7`, and inspect the output.

Our solution works to the extent of receiving HL7 v2.3.1 messages, and acknowledging them, transforming ADT messages to A08 for one stream of processing and writing them to files in the file system.

## Send invalid ADT messages

As before, we will use the CMDHL7Sender command line client to read a file containing a single HL7 ADT message, which is invalid, and submit it to the ADT Receiver endpoint. We will look at the exception file in our configured output directory – for me c:\hl7\received, and then review message tracking information in the Healthcare Integration Console.

☐ Copy the ADT_A01_output-1.hl7 file and give a copy a new name ADT_A01_broken.hl7

☐ Using a text editor which _does not_ change line endings, for example "Notepad++", edit the message and change the name of the EVN segment to read EVX instead. An example is shown below.

```
MSH|^~\&|SystemA|HosA|PI|MDM|2008090801529||ADT^A01|000000_CTLID_2008
090801529|P|2.3.1|||AL|NE
EVX|A01|2008090801529|||JavaCAPS6^^^^^^^USERS
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7
South 3rd Circle^^Downham Market^England -
Norfolk^30828^UK||||||||A2008090801529
PV1|1|I|||I|||FUL^Fulde^Gordian^^^^^^^^^^MAIN|||EMR|||||||||V200809080
1529^^^^VISIT|||||||||||||||||||||||||2008090801529
```

☐ In a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Sender_v0.7.jar -a SystemA -b HosA -c ID_
-n 1 -d \r\r\n -p 22222 -h localhost -t 30000 -f
c:\hl7\adt\sources\ADT_A01_broken_1.hl7
```

☐ Inspect the Negative ACK which the command line tool displays in the console window

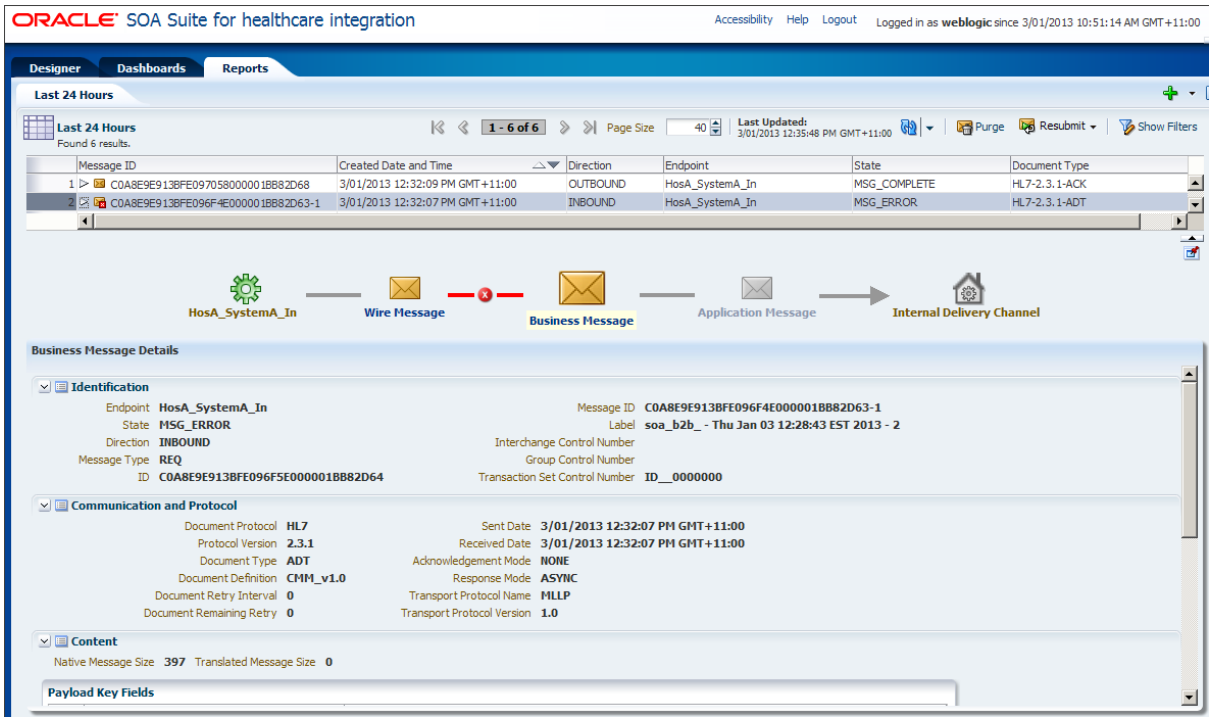
```
MSH|^~\&|PI|MDM|SystemA|HosA|20130103||ACK^A01|1001|P|2.3.1
MSA|AE|ID__0000000
ERR|^^^100&0x810070:Rejecting:Unrecognized data was found in the data
file. The last~^^^100&0x810070:Rejecting:Unrecognized data was found in
the data file. The last~EVN^3^^100&0x81004d:Rejecting:Segment EVN (EVN
- event type segment) at guidelin~PID^3^^100&0x81004d:Rejecting:Segment
PID (PID - patient identification segment)
```

☐ Inspect the content of the exception file, ipException_*nn*.xml – for me it looks like the text below

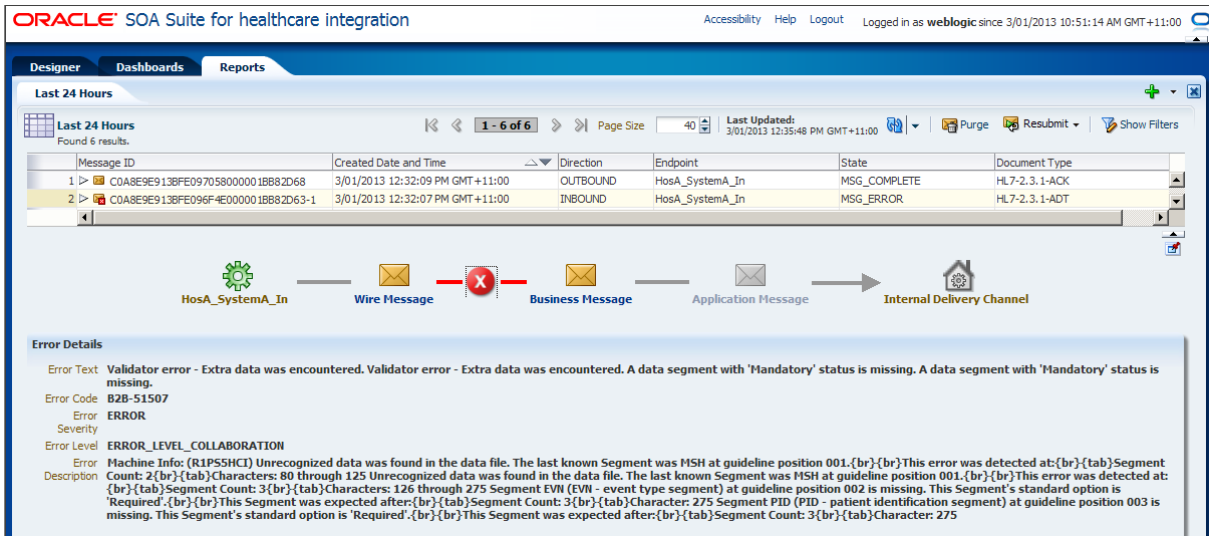
```
<?xml version="1.0" encoding="UTF-8" ?><Exception
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://integration.oracle.com/B2B/Exception">
  <correlationId>null</correlationId>
  <b2bMessageId>C0A8E9E913BFE096F4E000001BB82D63-1</b2bMessageId>
  <errorCode>B2B-51507</errorCode>
  <errorText>
  Validator error - Extra data was encountered.
Validator error - Extra data was encountered.
A data segment with 'Mandatory' status is missing.
A data segment with 'Mandatory' status is missing.


  </errorText>
  <errorDescription>
  Machine Info: (R1PS5HCI)
Unrecognized data was found in the data file. The last known Segment was MSH at
guideline position 001.{br}{br}This error was detected at:{br}{tab}Segment
Count: 2{br}{tab}Characters: 80 through 125
Unrecognized data was found in the data file. The last known Segment was MSH at
guideline position 001.{br}{br}This error was detected at:{br}{tab}Segment
Count: 3{br}{tab}Characters: 126 through 275
Segment EVN (EVN - event type segment) at guideline position 002 is missing. This
Segment's standard option is 'Required'.{br}{br}This Segment was expected
after:{br}{tab}Segment Count: 3{br}{tab}Character: 275
Segment PID (PID - patient identification segment) at guideline position 003 is
missing. This Segment's standard option is 'Required'.{br}{br}This Segment was
expected after:{br}{tab}Segment Count: 3{br}{tab}Character: 275


  </errorDescription>
  <errorSeverity>2</errorSeverity>
  <errorDetails>
          <parameter name="hc.messageId"
value="C0A8E9E913BFE096F4E000001BB82D63-1"/>
          <parameter name="hc.documentTypeName" value="ADT"/>
          <parameter name="hc.documentProtocolVersion" value="2.3.1"/>
          <parameter name="hc.documentDefinitionName" value="CMM_v1.0"/>
          <parameter name="hc.documentProtocolName" value="HL7"/>
```

```
          <parameter name="hc.messageType" value="1"/>
          <parameter name="hc.fromEndpoint" value="HosA_SystemA_In"/>
  </errorDetails>
</Exception>
```

☐ In the "Healthcare Integration Console" switch to "Reports" Tab and select the entry with the state of "MSG_ERROR"




☐ Click at the "Error" "button" between the "Wite Message" and the "Business Message" and inspect the exception




Our exercise if concluded. The solution will now process valid messages and will dump exceptions for invalid messages, or any other endpoint and HL7 processing in the "SOA Suite for healthcare integration" exceptions to files in the file systems. We could have gone to the trouble of parsing these messages, constructing custom notification messages, sending them via a Short Message Service or email, and so on. These are left as an exercise

and are examples of the kinds of things one can do in the Oracle SOA Suite, whether using the healthcare integration components or not.

## Summary

In this article we added exception handling to the routing and transformation solution developed in the article entitled "SOA Suite for healthcare integration Series – Domain Value Map (DVM) – On-the-fly Code Mapping", to explore how message processing exceptions can be redirected to a logic component for handling. The exception handling components will handle exceptions form all SOA Suite for healthcare integration applications.