Tivoli Netcool Support's Guide to the Message Bus Integration

by

Jim Hutchinson

Document release: 5.2

Table of Contents

1 Introduction

The Message Bus integration is composed of a number of common components and two integration products:

1. Message Bus Gateway [**nco_g_xml**]
2. Message Bus Probe [**nco_p_message_bus** formally nco_p_xml]

This document is intended to supplement the products documentation and provide guidance for specific integrations.

- Before reading this document it is recommended that the Message bus Probe and Gateway manuals are read.
- This document does not discuss the SCALA integration used with Log Analytics.
- There is a separate Support's guide for just the Webhook and WebSocket configuration for advanced usage.
- There is a separate Support's guide for just the Kafka configuration for advanced usage.

Note: The nco_p_message_bus probe replaced nco_p_xml probe.

1.1 Overview

This document is intended as a quick start to understanding the Message Bus gateway and probe

- Connection to the Apache ActiveMQ server [Transport JMS and MQTT]
- HTTP probe and gateway configuration [Transport HTTP]
  - HTTP for the message bus probe is the Webhook transport
- Other common probe examples
- General guidance
- Configuring the probe and gateway together

The files and example configurations provided allow the user to test the solution with minimal configuration.

The probe and gateway are Java based integrations that use the common-transformer and common-transport modules.

Gateway supporting packages:

- gateway-libngjava
- **common-transformer**
- **common-transportmodule**
- common-sslutility (not windows)

Probe supporting packages

- probe-nonnative-base
- probe-sdk-java
- **common-transformer**
- **common-transportmodule**

It is therefore recommended that the probe and gateway configuration files are copied to custom locations to ensure that they are not overwritten, and to allow easy comparison with the default files following any package updates.

1.2 Capabilities

The message bus probe and gateway share a common transport module that allows them to connect to various services. Along with the generic probe solution, there are a number of specific customisations of the probe provided with the message bus probe package. In addition to these, specific probes have been produced that use the transport and transformer modules, such as the CISCO APIC probe.

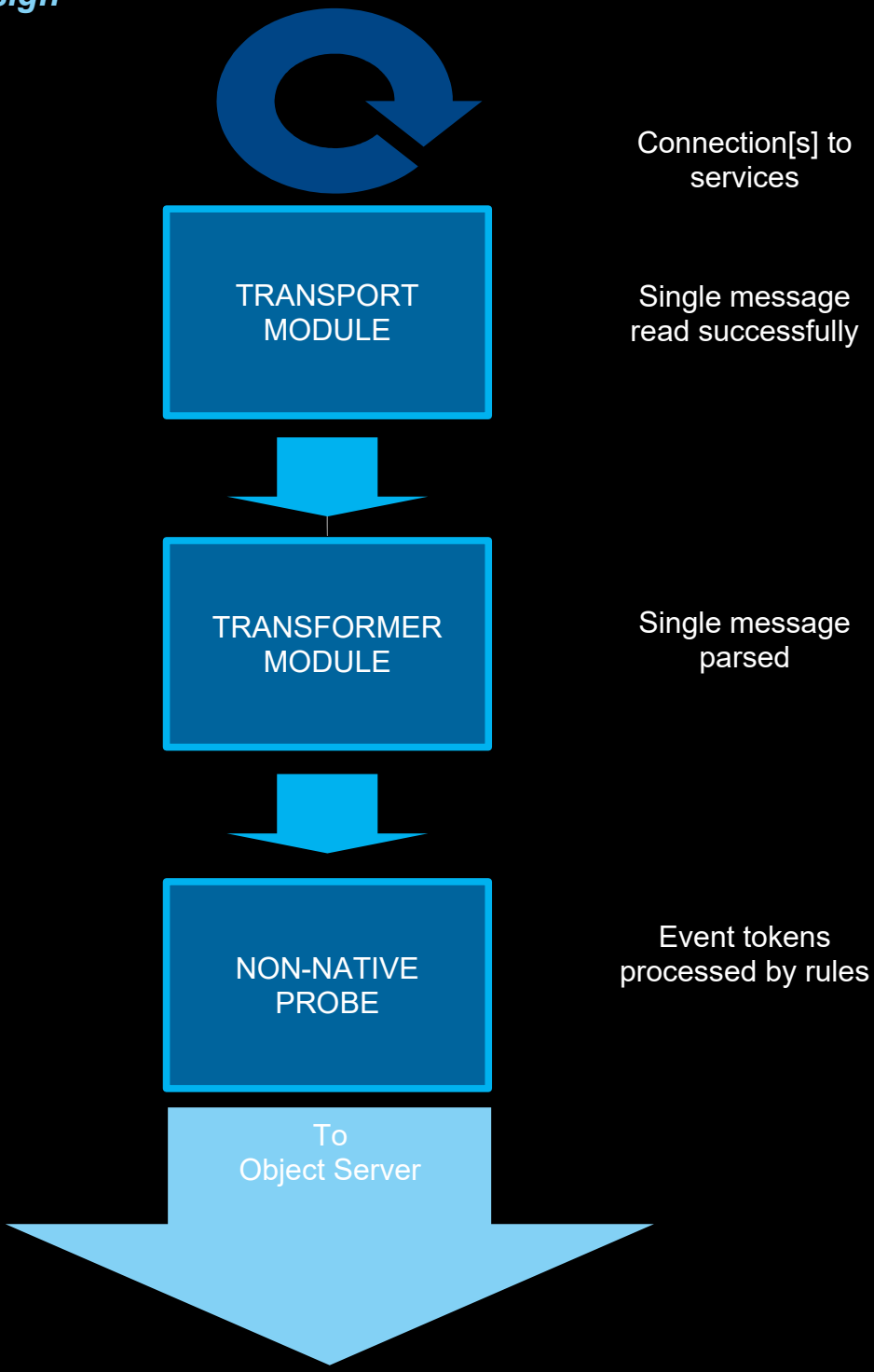
For the Generic message bus integration, the following capabilities are available:

	Probe	Gateway
XML data format	X	X
JSON data format	X	X
KAFKA	X	X
JMS	X	X
MQTT	X	X
Websocket	X	
Webhook [HTTP]	X	
Webhook2	X	
HTTP		X
FILE	X	X
SOCKET		X
Apache Pulsar	X	X
EventSource	X	
MultiChannelHttp		X
SCALA		X

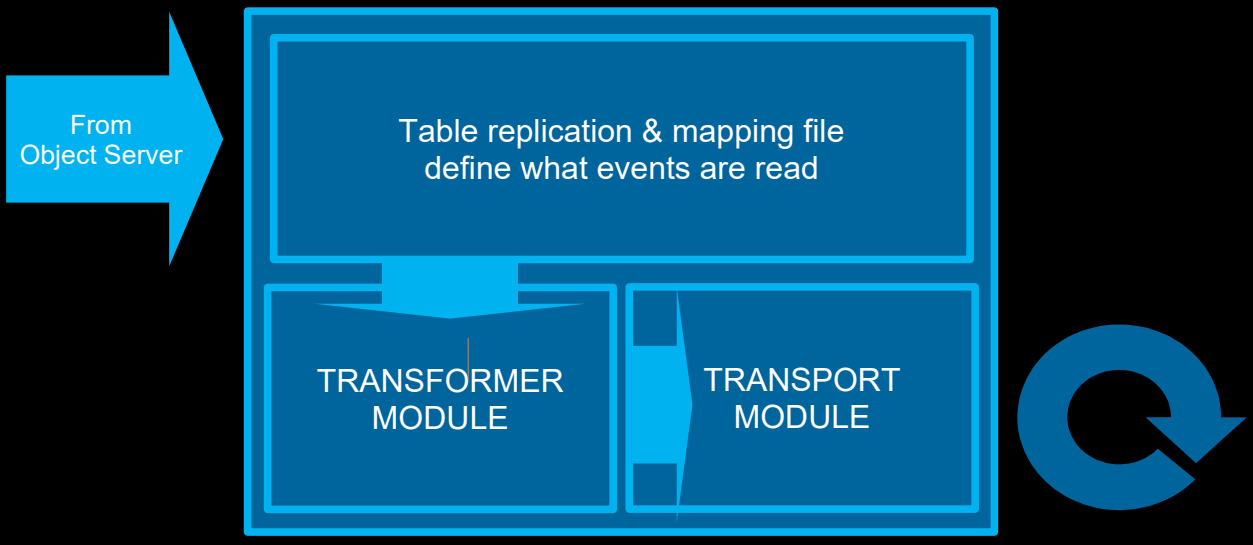
Notes:
The Message bus probe covers both XML and JSON messages.
The Message bus probe requires the property MessagePayload to be set to 'xml' for it to parse XML messages, which was the default behaviour of the nco_p_xml probe.

The Message bus [XML] Gateway uses a property to switch between XML and JSON message formats;
Gate.XMLGateway.TransformerInputType
The Message bus [XML] Gateway with HTTP transport types, requires bufferSize to be set to 1 to send XML events to the Message bus probe.

1.3 Probe design



1.4 Gateway design



2 JMS Transport with Apache MQ

2.1 Overview

The key configuration files are;

G_JMS.props or **message_bus.props** : integration properties
jmsTransport.properties : JMS Connection details
jndi.properties : JNDI translation [Apache ActiveMQ]
transformers.xml : Token translation

Apache ActiveMQ class files:

activemq-all-5.x.x.jar

etc.

2.2 Requirements

The Message Bus probe and gateway include a number of supporting features that may be updated outside of the probe or gateway packages. Check to see if there are updates to the following components after installing the probe or gateway:

common-transportmodule
common-transformer

For the Gateway only:
gateway-libngjava
gateway-libngtktk

The version of java used is determined using the wrapper scripts. It may be necessary to alter these scripts or the environment scripts to set the correct version of Java for the installation.

FQDN is used to define the MQ server; It stands for the Fully Qualified DomainName of the Apache ActiveMQ server, but can also be the servers IP Address. The important configuration is that the systems are able communicate via TCP using the given name or IP Address.

At the time of writing the wmqtt.jar file was not included with the transport module and needed to downloaded from the IBM support site separately, and is used when TransportType ‘MQTT’ is required.

2.3 Apache ActiveMQ server

The ActiveMQ product is a light-weight JMS server provided by Apache.

2.3.1 Downloading and installing

Download the product from the main web site;

http://activemq.apache.org/index.html

Install the product on an appropriate server.

2.3.2 Starting the ActiveMQ server

Configure the correct java environment for the version of ApacheMQ;

```
java -version
```

Start the main services

```
cd apache-activemq-5.13.0/bin
./activemq start
```

The administration server is started as;

```
./activemq-admin start
```

2.3.3 Create the default queue and topic

Using the correct hostname (localhost) login to the ActiveMQ server using a supported web browser;

http://localhost:8161/admin/

Note: Apache ActiveMQ servers can automatically create queues and topics.

2.4 Message Bus Gateway

2.4.1 Overview of configuration and jar files

$OMNIHOME/gates/G_JMS:

- **G_JMS.props**
- xml.map
- xml.reader.tblrep.def
- xml.startup.cmd

$OMNIHOME/gates/G_JMS/java

- **activemq-all-5.13.0.jar**
- **log4j-1.2.17.jar**
- **slf4j-log4j12-1.7.13.jar**
- **wmqtt.jar** [required for the MQTT transport method]

$OMNIHOME/gates/G_JMS/conf

- **jmsTransport.properties**
- **mqttTransport.properties**
- **jndi.properties**
- **transformers.xml**

$OMNIHOME/gates/java

- nco_g_xml.jar
- ngjava.jar

$OMNIHOME/java/conf

addnvpair.xsl
cbe2nvpairs.xsl
fileTransport.properties
mqttTransport.properties
netcool2cbe.xsl
netcool2nvpairs.xsl
netcool2wef.xsl
wbe2nvpairs.xsl
wbepl2nvpairs.xsl
wbm2nvpairs.xsl
wef2nvpairs.xsl

2.4.2 Setting the Java libraries

For the latest version of Netcool/OMNIbus the gateway may need to have the JRE_DIR defined to use the latest IBM Java.

If the Java library is not found the gateway reports:

error while loading shared libraries: libjvm.so: cannot open shared object file: No such file or directory

To modify the Java path:

e.g.

```
vi $NCHOME/omnibus/platform/linux2x86/bin/nco_g_xml.env

if [ -d ${BASE_JRE_DIR}/jre_1.6.7 ];
then
        JRE_DIR=${BASE_JRE_DIR}/jre_1.6.7
elif [ -d ${BASE_JRE_DIR}/jre_1.5.6 ];
then
        JRE_DIR=${BASE_JRE_DIR}/jre_1.5.6
…
:wq
```

2.4.3 Apache ActiveMQ CLASSPATH

The Apache ActiveMQ uses JNDI and requires the jndi.properties file to be in the CLASSPATH along with the required jar files.

e.g.

The jndi.properties directory is added to the jars classpath:-

```
setenv CLASSPATH $OMNIHOME/java/conf
```

The CLASSPATH with MQTT would be:

```
setenv CLASSPATH $OMNIHOME/gates/G_JMS/conf
setenv CLASSPATH $OMNIHOME/gates/G_JMS/java/activemq-all-5.13.0.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_JMS/java/log4j-1.2.17.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_JMS/java/slf4j-log4j12-1.7.13.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_JMS/java/wmqtt.jar:$CLASSPATH
#EOF
```

For production usage you can use the gateways env file or the property Gate.Java.ClassPath to set the CLASSPATH, otherwise use the users environment. Create a different user per gateway type if more than one CLASSPATH is required.

You can also explicitly set the classpath in the gateways property file:

```
Gate.Java.ClassPath                          : '$CLASSPATH'
```

2.4.4 Properties file

```
Name                                 : 'G_JMS'
MessageLog                           : '$OMNIHOME/log/G_JMS.log'
PropsFile                            : '$OMNIHOME/gates/G_JMS/G_JMS.props'
Gate.MapFile                         : '$OMNIHOME/gates/G_JMS/xml.map'
Gate.StartupCmdFile                  : '$OMNIHOME/gates/G_JMS/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile      : '$OMNIHOME/gates/G_JMS/xml.reader.tblrep.def'
Gate.XMLGateway.TransformerFile      : '$OMNIHOME/gates/G_JMS/conf/transformers.xml'
Gate.XMLGateway.TransportFile        : '$OMNIHOME/gates/G_JMS/conf/jmsTransport.properties'
Gate.XMLGateway.TransportType        : 'JMS'
Gate.XMLGateway.MessageID            : 'netcoolNVP'
#EOF
```

2.4.5 Transformers.xml file

The basic Transformers file would just contain the default transformer to the named topic '**netcool**' via the MessageID defined in the properties file [netcoolNVP]:-

```
        vi $OMNIHOME/gates/G_JMS/conf/transformers.xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Northbound (gateway) transformer definitions -->
        <tns:transformer name="netcoolNVP" type="northbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
        </tns:transformer>
</tns:transformers>
```

2.4.6 jmsTransport.properties

```
vi $OMNIHOME/gates/G_JMS/conf/jmsTransport.properties
initialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory
# JMS
providerURL=tcp://localhost:61616
# MQTT
#providerURL=tcp://localhost:1883
topicConnectionFactory=ConnectionFactory
#queueConnectionFactory=ConnectionFactory
topicName=netcool
#queueName=netcool
username=activemq
password=activemq
#EOF
```

2.4.7 jndi.properties

Note: The jndi.properties is not used in the current version of ApacheMQ but was used in earlier versions. It's configuration settings are included for completeness.

```
vi $OMNIOHME/gates/G_JMS/conf/jndi.properties
java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url=vm://FQDN
connectionFactoryNames=ConnectionFactory
#queue.netcool=netcool
topic.netcool=netcool
#EOF
```

2.5 Message Bus Probe

The Message Bus probe reads data from the queue and parses the data to the object servers.

In order to control the settings for the Message Bus probe, create an nco_p_message_bus probe directory in the $OMNIHOME/probes/java directory, adding copies of the required files;
e.g.
Copy files from the default location $OMNIHOME/java/conf to the probe specific configuration directory

- jndi.properties [needs to be created]
- jmsTransport.properties
- transformers.xml

Ensure that the required jar files [e.g. activemq-all-*.*.*.jar] are installed in the probe specific configuration directory so that the non-native probe environment script can detect them.

2.5.1 CLASSPATH

For the latest non-native probe the nco_p_message_bus.env script can be used to set the CLASSPATH;

```
vi $NCHOME/omnibus/probes/java/nco_p_message_bus.env
# Add on required jar files for Apache ActiveMQ
export jarfile
CLASSPATH=${OMNIHOME}/probes/java/${PROGRAM}:$CLASSPATH
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/log4j-1.2.17.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/slf4j-log4j12-1.7.13.jar
# Uncomment for the MQTT transport
#CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/wmqtt.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/activemq-all-5.13.0.jar
# DEBUGGING
echo "CLASSPATH= " $CLASSPATH
echo $CLASSPATH | awk -F\: '{for(i=1;i<=NF;i++)print $i}' | while read jarfile
do
echo $jarfile
done
#EOF
:wq
```

The probe specific directory $NCHOME/omnibus/probes/java/nco_p_message_bus will now contain;

- activemq-all-5.13.0.jar
- wmqtt.jar [if required]
- log4j-1.2.17.jar
- slf4j-log4j12-1.7.13.jar
- jmsTransport.properties
- transformers.xml
- jndi.properties

2.5.2 Message Bus probe properties file

```
# JMS Transport
TransportType                  : 'JMS'
MessagePayload                 : 'xml'
TransportFile                  : '$OMNIHOME/probes/java/nco_p_message_bus/jmsTransport.properties'
# MQTT Transport
#TransportType                 : 'MQTT'
#TransportFile                 : '$OMNIHOME/probes/java/nco_p_message_bus/mqttTransport.properties'
TransformerFile                : '$OMNIHOME/probes/java/nco_p_message_bus/transformers.xml'
#EOF
```

Note: **$OMNIHOME** should be expanded to the full path.

2.5.3 Transformers.xml file

The basic Transformers file just contains the default southbound transformer to the named topic ‘netcool’ as defined in the transport properties file:-

```
vi $OMNIHOME/probes/java/nco_p_message_bus/transformers.xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Southbound (probe) transformer defintions -->
<tns:transformer name="netcool2nvpairs" type="southbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTransformer">
<tns:property name="xsltFilename" type="java.lang.String" value="$OMNIHOME/java/conf/netcool2nvpairs.xsl"
description="XSLT file for converting Netcool events to name/value pairs"/>
</tns:transformer>
</tns:transformers>
```

2.5.4 jmsTransport.properties

```
vi $OMNIHOME/probes/java/nco_p_message_bus/jmsTransport.properties
initialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory
providerURL=tcp://FQDN:61616
queueName=netcool
queueConnectionFactory=ConnectionFactory
username=activemq
password=activemq
#EOF
```

2.5.5 jndi.properties

Note: The jndi.properties is not used in the current version of ApacheMQ but was used in earlier versions. It's configuration settings are included for completeness.

```
vi $OMNIHOME/probes/java/nco_p_message_bus/jndi.properties
java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url=vm://FQDN
connectionFactoryNames=ConnectionFactory
queue.netcool=netcool
#EOF
```

2.5.6 Windows Considerations

Windows does not allow the CLASSPATH to be set easily, therefore the nco_p_message_bus.bat file will need to be edited.

e.g.

```
set APACHEMQ_CLASSPATH=c:/apachemq/log4j-1.2.14.jar;c:/apachemq/slf4j-log4j12-1.5.11.jar;c:/apachemq/activemq-all-5.5.1.jar;c:/apachemq/

REM *** args to execute the probe

set PROGARGS=javaw -Xrs -cp %PROBE_CLASSPATH%;%NSPROBE_CLASSPATH%;%TRANSPORT_CLASSPATH%;%TRANSFORMER_CLASSPATH%;%APACHEMQ_CLASSPATH% nco_p_message_bus
```

Where the directory ‘c:/apachemq’ contains the required jar files for the Apache ActiveMQ server, jndi.properties and jmsTransport.properties.

2.5.7 Sending events

In the example configuration, events can be sent using the ‘Send’ feature available on the Apache ActiveMQ GUI. Start the probe from the command and login to the Apache ActiveMQ server. Check the number of consumers is ‘1’ in the Queue monitor tab window. Select the ‘Send’ tab, and enter ‘netcool’ as the Destination. Enter a valid XML string into the Message body section. Click ‘send’ and review the probes debug log file.

2.6 Message Bus Probe : MQTT example

The MQTT transport method can be used to provide an active alarm event stream. However, for the resynchronisation of events from the queue, the ActiveMQ server needs to be configured to provide a resend service.

2.6.1 message_bus.props

```
TransportType           : 'MQTT'
MessagePayload          : 'xml'
TransformerFile         : '$NCHOME/omnibus/probes/java/nco_p_message_bus/transformers.xml'
TransportFile           : '$NCHOME/omnibus/probes/java/nco_p_message_bus/mqttTransport.properties'
#EOF
```

2.6.2 mqttTransport.properties

In the example the MQTT service was added to run locally, so ‘localhost’ was used:

```
connectionURL=tcp://localhost:1883
clientId=MQTTClient
topicName=netcool
keepAlive=30
cleanStart=false
#EOF
```

When cleanStart Is set to false all the events on the topic are read since the client, given by the clientId, was last connected.

2.6.3 transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Southbound (probe) transformer defintions -->
        <tns:transformer name="netcool2nvpairs" type="southbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
                <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting Netcool events to name/value
pairs"/>
        </tns:transformer>
</tns:transformers>
#EOF
```

2.6.4 jndi.properties

The jndi.properties is not used in the current version of ApacheMQ but was used in earlier versions. It's configuration settings are included for completeness.

```
java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url=vm://FQDN
connectionFactoryNames=ConnectionFactory
topic.netcool=netcool
#EOF
```

2.6.5 Required jar files

The $NCHOME/omnibus/probes/java/nco_p_message_bus directory would contain the following files:

wmqtt.jar
activemq-all-5.7.0.jar
log4j-1.2.17.jar
slf4j-log4j12-1.6.6.jar

jndi.properties
mqttTransport.properties
transformers.xml

Where **wmqtt.jar** is the WebSphere MQTT client jar and needs to be added to the nco_p_message_bus.env file:

```
vi $NCHOME/omnibus/probes/java/nco_p_message_bus.env
# Add on required jar files for Apache ActiveMQ
CLASSPATH=${OMNIHOME}/probes/java/${PROGRAM}:$CLASSPATH
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/log4j-1.2.14.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/slf4j-log4j12-1.5.11.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/wmqtt.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/activemq-all-5.5.1.jar
echo "CLASSPATH= " $CLASSPATH
#EOF
:wq
```

3 HTTP Transport with XML

3.1 XML Gateway

3.1.1 G_HTTP.props

```
Gate.XMLGateway.TransportType    : 'HTTP'
Gate.Reader.Server               : 'NCOMS2'
Name                             : 'G_HTTP'
MessageLog                       : '$OMNIHOME/log/G_HTTP.log'
PropsFile                        : '$OMNIHOME/gates/G_HTTP/G_HTTP.props'
Gate.MapFile                     : '$OMNIHOME/gates/G_HTTP/xml.map'
Gate.StartupCmdFile              : '$OMNIHOME/gates/G_HTTP/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile  : '$OMNIHOME/gates/G_HTTP/xml.reader.tblrep.def'
Gate.XMLGateway.TransformerFile  : '$OMNIHOME/gates/G_HTTP/transformers.xml'
Gate.XMLGateway.TransportFile    : '$OMNIHOME/gates/G_HTTP/httpTransport.properties'
Gate.XMLGateway.MessageID        : 'netcoolEvents'
```

3.1.2 httpTransport.properties

The transport properties file only need to contain the client URL.

```
clientURL=http://localhost:12345
#EOF
```

3.1.3 transformers.xml

The transformers file only need to contain the northbound transformer, with the correct MessageID.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Northbound (gateway) transformer definitions -->
        <tns:transformer name="netcoolEvents" type="northbound" endpoint="http://localhost:12345"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
        </tns:transformer>
</tns:transformers>
```

3.2 Message Bus Probe

3.2.1 message_bus.props

```
Server            : 'NCOMS1'
TransportType     : 'HTTP'
MessagePayload    : 'xml'
TransformerFile   : '/opt/IBM/tivoli/netcool/omnibus/probes/nco_p_message_bus/transformers.xml'
TransportFile     : '/opt/IBM/tivoli/netcool/omnibus/probes/nco_p_message_bus/httpTransport.properties'
#EOF
```

3.2.2 httpTransport.properties

The Message Bus probe acts as a server and listens for HTTP messages on a given port, as defined in the transport properties using a URL type syntax:

```
serverPort=http:12345
#EOF
```

3.2.3 transformers.xml

The transformers only needs one southbound endpoint definition with the endpoint being defined as a URL type string as given in the transport properties.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://localhost"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Southbound (probe) transformer defintions -->
        <tns:transformer name="netcool2nvpairs" type="southbound" endpoint="http:12345"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
                <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting Netcool events to name/value
pairs"/>
        </tns:transformer>
</tns:transformers>
```

4 Probe Examples

4.1 WebSphere MQ

The following WebSphere example is provided as additional information to the Message Bus [XML] Probe manual.

4.1.1 WebSphere JNDI Binding

WebSphere uses the JNDI bindings directory to connect to the WebSphere MQ servers. In the jmsTransport.properties you can set this directory as the provider URL.
e.g.

```
providerURL=file:///opt/IBM/tivoli/netcool/omnibus/probes/java/nco_p_message_bus/jndi
```

Where the directory 'jndi' includes the .bindings JNDI file for the WebSphere servers the probe needs to connect to.

4.1.2 CLASSPATH

In order for the WebSphere MQ server to be accessed using WebSphere JNDI ensure the required jars are copied to the probe servers java directory:

For example in the $OMNIHOME/probes/java/nco_p_message_bus.env file extend the CLASSPATH to include the WebSphere MQ required jar files:

```
CLASSPATH=${OMNIHOME}/probes/java/${PROGRAM}:$CLASSPATH
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/providerutil.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/fscontext.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/dhbcore.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/jms.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/com.ibm.mq.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/com.ibm.mqjms.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/com.ibm.mq.jmqi.jar
#EOF
```

Obtaining the required JAR files:
You can ask the WebSphere Administrator for the file or use Fix Central.
File search : WS-MQ-Install-Java-All.jar
This JAR package needs to be unpacked and the required JARs placed in the probes directory.

4.1.3 Log file messages

When the probe cannot find the correct jar files it will log:

```
Debug: Creating initial context class using <given method names>
Information: Probewatch: Unable to get events;
Caused by: Cannot instantiate class: <FactoryName>
```

And

```
Debug: Creating initial context class using <given method names>
Information: Probewatch: Unable to get events; Caused by: javax.naming.Reference
```

When all the jar files are found but there is a problem with the specific queue the probe will log:

```
Debug: Creating initial context class using <given method names>
Debug: Looking up <queue> in JNDI
Debug: Creating queue connection using username and password
Information: [stderr] <specific class messages>
```

4.2 CISCO APIC v5.2

There is a CISCO APIC probe available, however, it cannot use the latest transport module. This is a problem where security is a concern, due to the log4j critical security alerts. The message bus probe release 19 can be configured to connect to the CISCO APIC v5.2 server.

4.2.1 Cisco APIC Property file

File : message_bus_cisco_apic.props

```
# Example message bus probe property settings for
# Cisco APIC v5.2
#
# Best practice
NetworkTimeout                  : 15
PollServer                      : 60
# Buffering
Buffering                       : 1
BufferSize                      : 200
FlushBufferInterval             : 11
# WebSocket Transport
TransportType                   : 'WebSocket'
# *** Set values
TransportFile                   : '$
{NCHOME}/omnibus/probes/<platform>/mb.ciscoApicTransport.properties'
Host                            : 'CISCO-HOST'
Port                            : 8080
Username                        : 'CISCO-USER'
Password                        : 'CISCO-PASSWORD'
#
# General values
#
HeartbeatInterval               : 0
MaxEventQueueSize               : 50000
TransportQueueSize              : 50000
ProbeWatchHeartbeatInterval     : 60
# Performance tuning
DisableDetails                  : 1
#
# SSL settings
# EnableSSL                       : 'true'
# KeyStore                        : '<full-path>/KeyStore.jks'
# KeyStorePassword                : 'netcool'

# Event parsing
#
MessagePayload                  : 'json.imdata.faultInst.attributes'
JsonMessageDepth                : 10
# EOF
```

4.2.2 Cisco APIC transport properties

```
File :  ${NCHOME}/omnibus/probes/<platform>/mb.ciscoApicTransport.properties

# Message bus probe release 19 transport properties
# for Cisco APIC v5.2
#
httpVersion=1.1
#responseTimeout=60
loginRequestURI=/api/aaaLogin.json
loginRequestMethod=POST
loginRequestContent="{\"aaaUser\" : {\"attributes\" : {\"name\" : \"++Username+
+\", \"pwd\" : \"++Password++\"}}}"
loginRefreshURI=/api/aaaRefresh.json
loginRefreshMethod=GET
loginRefreshContent=
loginRefreshInterval=180
logoutRequestURI=/api/aaaLogout.json
logoutRequestMethod=POST
logoutRequestContent="{\"aaaUser\" : {\"attributes\" : {\"name\" : \"++Username++\"}}}"
resyncRequestURI=/api/class/faultInst.json
resyncRequestMethod=GET
resyncRequestContent=
subscribeRequestURI=/api/class/faultInst.json?subscription=yes
subscribeRequestMethod=GET
subscribeRequestContent=
subscribeRefreshURI=/api/subscriptionRefresh.json?id=++subscriptionId++
subscribeRefreshMethod=GET
subscribeRefreshContent=
subscribeRefreshInterval=30
webSocketURI=/socket++APIC-cookie++
refreshRetryCount=3
#### - Use Message Bus probe to replace Cisco APIC ####
keepTokens=subscriptionId
responseTimeout=10
# EOF
```

4.3 Nokia OMS1350 example

This is an example of the property and properties file for connecting to the Nokia OMS1350 on the localhost using SSL on Linux using Netcool/OMNIbus 8.1.

4.3.1 message_bus_nokia_oms1350.props

```
# Object Server
Server                          : AGG_P
NetworkTimeout                  : 15
PollServer                      : 30
# Nokia OMS1350 settings
Host                            : 'localhost'
Port                            : 8443
MessagePayload                  : 'json'
TransportType                   : 'Cometd'
TransportFile                   : '${OMNIHOME}/probes/linux2x86/message_bus_nokia_oms1350.properties'
MessageLog                      : '$OMNIHOME/log/message_bus_nokia_oms1350.log'
RulesFile                       : '$OMNIHOME/probes/linux2x86/message_bus_nokia_oms1350.rules'
HeartbeatInterval               : 60
# Default processing
TransformerFile                 : ''
# Credentials to authenticate with system
Username                        : 'username'
Password                        : 'password'
# SSL configuration
EnableSSL                       : 'true'
KeyStore                        : '/tmp/netcool.jks'
KeyStorePassword                : 'netcool'
#EOF
```

4.3.2 message_bus_nokia_oms1350.properties

The default file nokiaOms1350CometdTransport.properties is found in $OMNIHOME/java/conf.

For this configuration HTTP needs to be set to https, with the other settings coming from the probes property file using the ++variable++ syntax. The version of Java used by the probe and the TLS version needs to match the servers for SSL to work correctly.

```
bayeuxClientTransport=long-polling

subscriptionChannel=/oms1350/events/otn/rest/alarmEvent

# SSL so use HTTPS
loginRequests={"requests":[{"uri":"/cas/v1/tickets","method":"POST","header":{"Content-Type":"application/x-www-
form-urlencoded","Authorization":"Basic ++Username++:++Password++"},"data":"username=++Username++&password=+
+Password++"},{"uri":"++Location++","method":"POST","data":"service=https://++Host++:8443/oms1350","header":
{"Content-Type":"application/x-www-form-urlencoded","Connection":"Keep-Alive"}},
{"uri":"/oms1350/data/plat/session/login","method":"POST","data":"user=++Username++&password=++Password+
+&ticket=++TICKET++","header":{"Content-Type":"application/x-www-form-urlencoded","Connection":"Keep-Alive"}}]}

securityProtocol=TLSv1

connectTimeout=5

httpHeaders={"Accept":"application/json","User-Agent":"Netcool/OMNIbus Message Bus Probe"}
```

4.3.3 Testing

It is possible to test the connectivity using netcat, other with SSL enabled the information sent is encrypted.
e.g.

```
nc -l -p 8443
```

4.4 Apache Kafka Example

The example provided uses the standard installation of the Apache Kafka server and the April 2018 release of the Message Bus probe. In order to make maintenance of the probes files easier, these are best copied to a common directory, in this example the probes/platform directory. The examples uses Netcool/OMNIbus 8.1 on Red Hat linux, whose platform directory is linux2x86.

4.4.1 Apache Kafka

The Apache Kafka server is available for download and can be installed on a system for testing.
It can be easily configured provided it has enough resources, and a supported version of Java.

4.4.2 User environment

The user running the Apache Kafka requires access to the ports and a supported version of Java.

Default ports used:

- 2181
- 9092

User environment setting:

```
csh
set path=(/opt/jdk1.8.0_144/jre/bin $path)
rehash
setenv LC_ALL "C"
setenv LANG "C"
```

4.4.3 Apache Kafka Server

The software can be downloaded as a GNU compressed tar file which can be extracted into a directory.

The Apache Kafka website includes a quick start guide:
https://kafka.apache.org/quickstart

Once unpacked the server can be started using the following commands:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

To prepare for the probe connection create the example topics:

- topicABC
- topicXYZ

Using the commands:

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic topicABC
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic topicXYZ
```

To check the topics are available use:

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

4.4.4 message_bus_kafka.props

```
#==========================================================================
# SETTING PROBE LOGS, PROPS, RULES
#==========================================================================
Manager                         : 'Kafka'
MessageLog                      : '$OMNIHOME/log/message_bus_kafka.log'
RulesFile                       : '$OMNIHOME/probes/linux2x86/message_bus_kafka.rules'

#==========================================================================
# SETTING TRANSPORT TYPE
#==========================================================================
TransportType                   : 'KAFKA'
#TransportFile                   : '$OMNIHOME/java/conf/kafkaTransport.properties'
TransportFile                   : '$OMNIHOME/probes/linux2x86/kafka/kafkaTransport.properties'

#==========================================================================
# SETTING PARSER CONFIGURATIONS. (SUPPORTS JSON OR XML)
#==========================================================================
# FOR PARSING JSON DATA
TransformerFile             : '$OMNIHOME/probes/linux2x86/kafka/message_bus_parser_config.json'
MessagePayload              : 'JSON'

# FOR PARSING XML DATA
# TransformerFile             : '$OMNIHOME/java/conf/transformers.xml'
# MessagePayload              : 'XML'

#PROTECTED REGION END#
```

4.4.5 kafkaTransport.properties

```
kafkaClientMode=CONSUMER
connectionPropertiesFile=$OMNIHOME/probes/linux2x86/kafka/kafkaConnectionProperties.json
```

4.4.6 kafkaConnectionProperties.json

```
{
    "zookeeper_client" :
        {
            "target" : "<FQDN>:2181",
            "properties" :
"/opt/nrv81/IBM/tivoli/netcool/omnibus/probes/linux2x86/kafka/zookeeperClient.properties",
            "java_sys_props" : "",
            "topic_watch": true,
            "broker_watch": true
        },
    "brokers" : "PLAINTEXT://<FQDN>:9092",
    "topics": "topicABC,topicXYZ",
    "kafka_client" :
        {
            "properties" :
"/opt/nrv81/IBM/tivoli/netcool/omnibus/probes/linux2x86/kafka/kafkaClient.properties",
            "java_sys_props" : ""
        }
}
```

4.4.7 zookeeperClient.properties

This can be an empty file.

4.4.8 kafkaClient.properties

```
acks=all
client.id=KafkaExampleProducer
group.id=KafkaConsumerGroupName
pollInterval=1
```

4.4.9 Sending events

Sending events to the probe from the Apache Kafka server.

Once the probe is connected JSON events can be sent through using the command line interface:

```
csh
set path=(/opt/jdk1.8.0_144/jre/bin $path)
rehash

bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topicABC
>
```

```
{"eventfactory":[{"subscription-id":"1234567890","resource-id":"aabbcc-ffff-0001-1000-abcdef000000","monitoring-
id":123456789012345678,"fault-type-id":123456789012345678,"fault-type-name":"CPU
Failure","severities":"major"}]}
```

Expected probe logging at message level debug:

```
Received message with length of 225 from endpoint topicABC: {"eventfactory":[{"subscription-
id":"1234567890","resource-id":"aabbcc-ffff-0001-1000-abcdef000000","monitoring-id":123456789012345678,"fault-
type-id":123456789012345678,"fault-type-name":"CPU Failure","severities":"major"}]}
Start parsing JSON message
No exact parser for endpoint topicABC found. Parser configuration for soure type "ANY" will be used.
Adding event of size 7 to queue
Done parsing JSON message
[Event Processor] resync_event:      false
[Event Processor] eventfactory.0.subscription-id:    1234567890
[Event Processor] eventfactory.0.fault-type-id:      123456789012345678
[Event Processor] eventfactory.0.monitoring-id:      123456789012345678
[Event Processor] eventfactory.0.fault-type-name:    CPU Failure
[Event Processor] eventfactory.0.severities: major
[Event Processor] eventfactory.0.resource-id:        aabbcc-ffff-0001-1000-abcdef000000
[Event Processor] Processing alert {0 remaining}
```

4.5 Logstash Webhook with SSL example

The platform used was linux as this include openssl.

4.5.1 Creating the SSL Certificates

Setting the Java path

```
cd $NCHOME
find . -name keytool
./IBM/tivoli/netcool/platform/linux2x86/jre64_1.7.0/jre/bin/keytool

csh
set path=($NCHOME/platform/linux2x86/jre64_1.7.0/jre/bin $path)
rehash
java -version
```

Creating the probe store

```
keytool -genkey -alias <FQDN> -keystore LogStashProbeStore.jks -keyalg RSA -sigalg
SHA1withRSA -storepass netcool

  [Unknown]:  <FQDN>
What is the name of your organizational unit?
  [Unknown]:  Support
What is the name of your organization?
  [Unknown]:  IBM
What is the name of your City or Locality?
  [Unknown]:  New York
What is the name of your State or Province?
  [Unknown]:  New York
What is the two-letter country code for this unit?
  [Unknown]:  US

keytool -export -alias <FQDN> -file probe.cer -keystore LogStashProbeStore.jks -storepass
netcool
```

Create a local CA

```
setenv RANDFILE rand
openssl req -new -keyout cakey.pem -out careq.pem
openssl x509 -signkey cakey.pem -req -days 3650 -in careq.pem -out caroot.cer -extensions
v3_ca
keytool -printcert -v -file caroot.cer
keytool -certreq -alias <FQDN> -keystore LogStashProbeStore.jks -file ProbeReq.csr

echo 1234>serial.txt
openssl x509 -CA caroot.cer -CAkey cakey.pem -CAserial serial.txt -req -in ProbeReq.csr
-out ProbeCA.cer -days 365


keytool -import -keystore LogStashProbeStore.jks -alias rootca -file caroot.cer
-storepass netcool
keytool -import -keystore LogStashProbeStore.jks -alias <FQDN> -file ProbeCA.cer
-storepass netcool


keytool -list -keystore LogStashProbeStore.jks -storepass netcool

Your keystore contains 2 entries
rootca, Jul 20, 2018, trustedCertEntry,
<FQDN>, Jul 20, 2018, keyEntry,
```

Directory : $NCHOME/omnibus/probes/linux2x86
File : message_bus.props-logstash-ssl

```
Server                          : 'AGG_P'
ServerBackup                    : 'AGG_B'
NetworkTimeout                  : 15
PollServer                      : 60
TransformerFile                 : '$
{OMNIHOME}/probes/linux2x86/message_bus_logstash/message_bus_logstash_parser.json'
TransportFile                   : '$
{OMNIHOME}/probes/linux2x86/message_bus_logstash/logstashWebhookTransport-ssl.properties'
TransportType                   : 'Webhook'
Port                            : 8888
HeartbeatInterval               : 60
EnableSSL                       : 'true'
KeyStore                        : '/tmp/LogStashProbeStore.jks'
KeyStorePassword                : 'netcool'
```

Directory : $NCHOME/omnibus/probes/linux2x86/message_bus_logstash
File : logstashWebhookTransport-ssl.properties

```
webhookURI=/probe/webhook/logstash
idleTimeout=180
```

Sending a test message

```
curl --cacert ./caroot.cer -v -H "Content-Type: application/json" -d '{"test":"Message
text"}' -X POST https://<FQDN>:8888/probe/webhook/logstash
```

Sending an event with some details:

```
curl --cacert ./caroot.cer -v -H "Content-Type: application/json" -d '{ "NodeName":
"localhost", "SourceName": "Commandline CURL", "Message": "This is a CURL test message",
"Severity":"informational", "Type":"Event"}' -X POST
https://<FQDN>:8888/probe/webhook/logstash
```

Extra features:
Directory : $NCHOME/omnibus/probes/java
File : nco_p_message_bus.env
Enable SSL logging to standard out and enable SSLv3:

```
# SSL Logging and enable SSLv3
# NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=ssl:handshake:verbose
-Dcom.ibm.jsse2.disableSSLv3=false $NCO_JPROBE_JAVA_FLAGS"
# Just SSL logging
NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=ssl:handshake:verbose $NCO_JPROBE_JAVA_FLAGS"
echo "NCO_JPROBE_JAVA_FLAGS=$NCO_JPROBE_JAVA_FLAGS"
```

Example command line usage with SSL logging enabled:

```
cd $NCHOME/omnibus/probes/linux2x86
../nco_p_message_bus -propsfile ./message_bus.props-logstash-ssl -messagelevel debug > $
{OMNIHOME}/log/message_bus-ssl.log
```

4.6 HTTP BasicAuth example

It is possible to protect the HTTP port using Basic Authentication.

File : $NCHOME/omnibus/probes/linux2x86/message_bus_HttpJsonBasicauth.props

```
# Object Server connection
Server          : 'AGG_P'
ServerBackup    : 'AGG_B'
NetworkTimeout  : 15
PollServer      : 60
# Tranport configuration
TransportType   : 'HTTP'
MessagePayload  : 'json'
TransformerFile : ''
TransportFile   : '$NCHOME/omnibus/probes/linux2x86/message_bus_HttpJsonBasicauthTransport.properties'
HeartbeatInterval : 60
ProbeWatchHeartbeatInterval : 60
```

File : $NCHOME/omnibus/probes/linux2x86/message_bus_HttpJsonBasicauthTransport.properties

```
serverPort=http:9191
username=username
password=password
#EOF
```

You can test the probe using curl:
e.g.

```
curl -v -H "Content-Type: application/json" -H "Authorization: Basic
dXNlcm5hbWU6cGFzc3dvcmQ=" -d '{"test":"Message"}' -X POST http://localhost:9191
```

Where the user and password string used is created using base64.
e.g.

```
echo -n "username:password" | base64
dXNlcm5hbWU6cGFzc3dvcmQ=
```

Example debug logging

```
HttpRequestHandler]: Connection from client 'localhost' (51704) to server port
'http:9191'.
[HttpParser]: Successfully read HTTP header.
[HttpParser]: Reading payload size = 18
[HttpParser]: Successfully read HTTP payload.
Performing JSON format check
JSON format OK
[HttpRequestHandler]: Adding posted payload from endpoint 'http:9191' to queue.
[{"test":"Message"}].
[HttpRequestHandler]: Sent response [200]
[Event Processor] resync_event:      false
[Event Processor] MESSAGE.META.Content-Length:       [18]
[Event Processor] MESSAGE.META.Accept:       [*/*]
[Event Processor] test:      Message
[Event Processor] MESSAGE.META.Content-Type: [application/json]
[Event Processor] MESSAGE.META.Authorization:        [Basic dXNlcm5hbWU6cGFzc3dvcmQ=]
[Event Processor] Processing alert {0 remaining}
```

4.7 KAFKA with TLS/SSL Example

The Message bus probe can connect to a KAFKA server running TLS/SSL for a more secure configuration. Given the KAFKA server is installed and configured for use already, the following example provides a typical TLS/SSL configuration.

4.7.1 Probe Property file

```
# Best practice settings
Server                  : 'AGG_P'
ServerBackup            : 'AGG_B'
NetworkTimeout          : 15
PollServer              : 60
# Buffering
Buffering               : 1
BufferSize              : 200
FlushBufferInterval     : 9
# General settings
MessageLevel            : 'debug'
Manager                 : 'Kafka-SSL'
MessageLog              : '$NCHOME/omnibus/log/message_bus_kafka_ssl.log'
RulesFile               : '$NCHOME/omnibus/probes/linux2x86/message_bus_kafka.rules'
# KAFKA TLS/SSL configuration
TransportType           : 'KAFKA'
TransportFile           : '$NCHOME/omnibus/probes/linux2x86/kafka-ssl/kafkaTransport.properties'
# Event parsing
MessagePayload          : 'JSON'
TransformerFile         : '$NCHOME/omnibus/probes/linux2x86/kafka-ssl/message_bus_parser_config.json'
# Other settings
HeartbeatInterval       : 60
InitialResync           : 'false'
StreamCapture           : 'false'
StreamCaptureFile       : '$NCHOME/omnibus/var/message_bus_kafka_maguila.stream'
#EOF
```

4.7.2 kafkaTransport.properties

```
#================================================================================
# KAFKA CLIENT MODE
# VALID VALUES = CONSUMER | PRODUCER
#================================================================================
kafkaClientMode=CONSUMER
#================================================================================
# LOCATION OF FILE CONTAINING KAFKA CONNECTION PROPERTIES
#================================================================================
connectionPropertiesFile=$OMNIHOME/probes/linux2x86/kafka-ssl/kafkaConnectionProperties.json
#EOF
```

4.7.3 kafkaConnectionProperties.json

Note: The probe server needs to be able to resolve the servers FQDN and hostname.

```
{
    "zookeeper_client" :
        {
            "target" : "",
            "properties" : "",
            "java_sys_props" : "",
            "topic_watch": true,
            "broker_watch": true
        },
    "brokers" : "SSL://<FQDN>:9093",
    "topics": "topicABC,topicXYZ,heartbeating",
    "kafka_client" :
        {
            "properties" : "/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/kafka-ssl/kafkaClient.properties",
            "java_sys_props" : ""
        }
}
```

4.7.4 kafkaClient.properties

```
# SSL Activiation
security.protocol=SSL
sasl.mechanism=PLAIN
ssl.protocol=TLSv1.2
ssl.enabled.protocols=TLSv1.2
# Java Keystore
ssl.keystore.location=/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/kafka-ssl/SSL/KafkaProbeKeyStore.jks
ssl.keystore.password=netcool
ssl.keystore.type=JKS
ssl.key.password=netcool
ssl.client.auth=required
# Java Truststore
ssl.truststore.location=/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/kafka-ssl/SSL/KafkaServerTrustStore.jks
ssl.truststore.password=netcool
ssl.truststore.type=JKS
# Group and client settings
group.id=KafkaMBProbeGroup001
client.id=KafkaMBProbe001
# Other settings
pollInterval=1000
max.poll.records=1000
enable.auto.commit=true
auto.commit.interval.ms=5000
acks=all
max.block.ms=60000
retries=0
#EOF
```

4.7.5 Referenced files

In order to maintained a easily administered set of files is best to create an TLS/SSL integrations directory to hold the probes configuration files, and the SSL certificate files.

Directory : $NCHOME/omnibus/probes/linux2x86/kafka-ssl
This directory is used tohold the probes configuration files.

- kafkaClient.properties
- kafkaConnectionProperties.json
- kafkaTransport.properties
- message_bus_parser_config.json

Directory : $NCHOME/omnibus/probes/linux2x86/kafka-ssl/SSL
This directory is used to manage the ROOT CA and to create the JKS files.

- KafkaProbeKeyStore.jks
- KafkaServerKeyStore.jks
- KafkaServerTrustStore.jks
- cakey.pem
- careq.pem
- caroot.cer
- caroot.srl
- localhost.signed.cert
- localhost.signed.crt
- localhost.unsigned.cert
- localhost.unsigned.crt
- rand

4.7.6 Creating the TLS/SSL Certificates

Set a common Java Home path before starting

```
set path=(/opt/ibm-java-x86_64-80/jre/bin $path)
rehash
```

ROOT CA creation

```
setenv RANDFILE rand
openssl req -new -keyout cakey.pem -out careq.pem
openssl x509 -signkey cakey.pem -req -days 3650 -in careq.pem -out caroot.cer -extensions
v3_ca
```

Created files:

- caroot.cer
- rand
- cakey.pem
- careq.pem

Checking files using keytool

```
keytool -printcert -v -file caroot.cer
keytool -list -keystore KafkaServerKeyStore.jks -storepass netcool
keytool -list -keystore KafkaServerTrustStore.jks -storepass netcool
keytool -list -keystore KafkaProbeKeyStore.jks -storepass netcool
```

Create the KAFKA Servers Trust Store

```
keytool -keystore KafkaServerTrustStore.jks -alias ROOTCA -import -file caroot.cer
Enter keystore password: netcool
Re-enter new password: netcool
```

e.g.

```
Owner: EMAILADDRESS=root@<PROBE-FQDN>, CN=<PROBE-FQDN>, OU=Netcool, O=IBM, L=New York,
ST=New York, C=US
Issuer: EMAILADDRESS=root@<PROBE-FQDN>, CN=<PROBE-FQDN>, OU=Netcool, O=IBM, L=New York,
ST=New York, C=US
```

Create the KAFKA Servers Key Store

```
keytool -keystore KafkaServerKeyStore.jks -alias localhost -validity 365 \
-genkey -keyalg RSA -ext SAN=DNS:<KAFKA-FQDN>
Enter keystore password: netcool
Re-enter new password: netcool
What is your first and last name? <KAFKA-FQDN>
What is the name of your organizational unit? Netcool
What is the name of your organization? IBM
What is the name of your City or Locality? New York
What is the name of your State or Province? New York
What is the two-letter country code for this unit? US

keytool -keystore KafkaServerKeyStore.jks -alias localhost \
-certreq -file localhost.unsigned.crt

openssl x509 -req -CA caroot.cer -CAkey cakey.pem -in localhost.unsigned.crt \
-out localhost.signed.crt -days 3650 -CAcreateserial

keytool -keystore KafkaServerKeyStore.jks -alias ROOTCA -import -file caroot.cer
Enter keystore password: netcool

keytool -keystore KafkaServerKeyStore.jks -alias localhost \
-import -file localhost.signed.crt
Enter keystore password: netcool
```

Create the Probes Key Store

```
keytool -keystore KafkaProbeKeyStore.jks -alias localhost -validity 3650 -genkey -keyalg
RSA -ext SAN=DNS:<PROBE-FQDN>

keytool -keystore KafkaProbeKeyStore.jks -alias localhost -certreq -file
probehost.unsigned.cert

openssl x509 -req -CA caroot.cer -CAkey cakey.pem \
-in probehost.unsigned.cert \
-out probehost.signed.cert \
-days 3650 -CAcreateserial

keytool -keystore KafkaProbeKeyStore.jks -alias ROOTCA -import \
-file caroot.cer

keytool -keystore KafkaProbeKeyStore.jks -alias localhost -import \
-file probehost.signed.cert
```

4.7.7 KAFKA Server configuration

File : config/server.properties

Standard configuration plus:

```
# Following SSL configs are needed
listeners=PLAINTEXT://:9092,SSL://:9093
ssl.keystore.location=/opt/kafka_2.12-1.1.0/var/KafkaServerKeyStore.jks
ssl.keystore.password=netcool
ssl.key.password=netcool
ssl.truststore.location=/opt/kafka_2.12-1.1.0/var/KafkaServerTrustStore.jks
ssl.truststore.password=netcool
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
ssl.client.auth=required
#EOF
```

Running The KAFKA Server

Zookeeper server:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

KAFKA server:

```
bin/kafka-server-start.sh config/server.properties
```

Checking the TLS/SSL Service is running:

```
openssl s_client -debug -connect localhost:9093 -tls1
```

File : heartbeat_ssl.sh

```
#! /bin/sh
export DATE
DATE=`date +%Y%m%d%H%M%S`
echo $DATE
bin/kafka-console-producer.sh --producer.config config/ssl-client.properties --broker-
list localhost:9093 --topic heartbeating << EOF
{"eventfactory":[{"heartbeat-id":"$
{DATE}","timestamp":"date","summary":"Heartbeat","severity":"information"}]}

EOF
#EOF
```

File : config/ssl-client.properties

```
security.protocol=SSL
ssl.truststore.location=/opt/kafka_2.12-1.1.0/var/KafkaServerTrustStore.jks
ssl.truststore.password=netcool
ssl.keystore.location=/opt/kafka_2.12-1.1.0/var/KafkaServerKeyStore.jks
ssl.keystore.password=netcool
ssl.key.password=netcool
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
ssl.client.auth=required
#EOF
```

You can check the heartbeats are sent using a local consumer

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9093  --topic heartbeating \
--consumer.config config/ssl-client.properties -from-beginning
```

5 Event parsing

The probe and gateway event parser is designed to support both JSON and XML.
With the message bus probe, the event data can be inspected using the streamcapture file, or for short events, from the endpoint debug logging.

Probe property settings.

```
TransformerFile                 : '${OMNIHOME}/java/conf/transformers.xml'
MessagePayload                  : 'xml|json'
MessageHeader                   : ''
JsonNestedPayload               : ''
JsonNestedHeader                : ''
JsonMessageDepth                : 3
RecordData                      : ''
```

The message bus probe is required to read the incoming event data, and format it into the Netcool event syntax.

Gateway property settings.

```
Gate.XMLGateway.TransformerFile       : '$OMNIHOME/java/conf/transformers.xml'
Gate.XMLGateway.TransformerInputType  : 'xml|json'
```

The XML gateway can be configured to meet the target systems event requirements, provided the target system accepts single events.

5.1 XML Events

5.1.1 Simple XML data

Refer to online references for acceptable XML data syntax, such as www.w3schools.com.

An example of XML data is.

```
<root>
  <child>
    <subchild>data</subchild>
  </child>
</root>
```

5.1.2 Example Netcool XML event

An event that can be parsed by the netcool2nvpairs.xsl file is.

```
<tns:netcoolEvent type="insert" xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool">
    <tns:netcoolField type="string" name="Identifier">MTTRAPD-EVENT-0001</tns:netcoolField>
    <tns:netcoolField type="string" name="TTNumber">TT00001</tns:netcoolField>
    <tns:netcoolField type="string" name="TTStatus">opened</tns:netcoolField>
    <tns:netcoolField type="integer" name="Class">123456</tns:netcoolField>
    <tns:netcoolField type="string" name="Summary">Summary Text MTTrapd</tns:netcoolField>
    <tns:netcoolField type="string" name="AlarmID">ABC123</tns:netcoolField>
    <tns:netcoolField type="string" name="Node">Node001</tns:netcoolField>
    <tns:netcoolField type="string" name="AlertGroup">test</tns:netcoolField>
    <tns:netcoolField type="string" name="AlertKey">xml</tns:netcoolField>
</tns:netcoolEvent>
```

The XML event source must send only one event per data push, as the probe only processes one data path per push.

5.1.3 Example XSL checker script

Directory : $NCHOME/omnibus/utils

File : XLSChecker.sh

```
#! /bin/sh
#
# xlschecker.sh [XSL] [XML]
#
# Uses the Netcool/OMNIbus environment to check
# XSL formatting of XML data

# Parse variables
if [ $# -ne 2 ]
then
  echo "Usage : `basename $0` [XSL] [XML]"
  exit
fi
# Check for input files
if [ ! -f $1 ]
then
  echo "Cannot locate file " $1
  exit
fi
if [ ! -f $2 ]
then
  echo "Cannot locate file " $2
  exit
fi
# Need to have NCHOME set
if [ -z "$NCHOME" ]
then
  echo "NCHOME required for script to work"
  exit
else
# Use the Probes 64 bit  Java
PATH=$NCHOME/platform/linux2x86/jre64_1.8.0/jre/bin:$PATH
fi
# Check for JARS
if [ ! -f $NCHOME/omnibus/java/jars/Transformer.jar ]
then
  echo "Cannot locate file " $NCHOME/omnibus/java/jars/Transformer.jar
  exit
fi
# Run the XSL checker
java -cp $NCHOME/omnibus/java/jars/Transformer.jar
com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer $1 $2
# EOF
```

5.1.4 Example XSL parsing

Create the example event input XML file, **example_netcool_event.xml**, from the event given earlier in the document.
Use this and the **netcool2nvpairs.xsl** to demonstrate how the XSL checker script is used to parse events from the command line. Note that the probe only parses one event at a time, so any custom format must have single events, not batches of events.

```
./XLSChecker.sh $NCHOME/omnibus/java/conf/default/netcool2nvpairs.xsl
/tmp/example_netcool_event.xml
Output from applying transformer
'/opt/IBM/tivoli/netcool/omnibus/java/conf/default/netcool2nvpairs.xsl' to source
file
'/tmp/example_netcool_event.xml':-

NetcoolEventAction:string:insert
Identifier:string:"MTTRAPD-EVENT-0001"
TTNumber:string:"TT00001"
TTStatus:string:"opened"
Class:integer:"123456"
Summary:string:"Summary Text MTTrapd"
AlarmID:string:"ABC123"
Node:string:"Node001"
AlertGroup:string:"test"
AlertKey:string:"xml"
```

Which confirms that the netcool2nvpairs.xsl XSL file is suitable for use in the transformers.xml file:

transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Southbound (probe) transformer defintions -->
        <tns:transformer name="netcool2nvpairs" type="southbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
                <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting Netcool
events to name/value pairs"/>
        </tns:transformer>
</tns:transformers>
```

5.1.5 Example custom XML event parsing

For XML event data it is best to keep the events simple and without any depth.

```
File : custom_example.xml

<?xml version="1.0" encoding="UTF-8"?>
<event>
   <id>xxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx</id>
   <version>1.0</version>
   <sequence_number>1234567890</sequence_number>
   <summary>WebserviceEvent</summary>
</event>


File : custom_example.xsl

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<xsl:for-each select="event">
<xsl:text>
</xsl:text>
<xsl:text>id:string:"</xsl:text>
<xsl:value-of select="id"/>"
<xsl:text>version:string:"</xsl:text>
<xsl:value-of select="version"/>"
<xsl:text>sequence_number:string:"</xsl:text>
<xsl:value-of select="sequence_number"/>"
<xsl:text>summary:string:"</xsl:text>
<xsl:value-of select="summary"/>"
</xsl:for-each>

</xsl:template>
</xsl:stylesheet>


./xslchecker.sh custom_example.xsl custom_example.xml
Output from applying transformer 'custom_example.xsl' to source file
'custom_example.xml':-
<?xml version="1.0" encoding="UTF-8"?>
id:string:"xxxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"
version:string:"1.0"
sequence_number:string:"1234567890"
summary:string:"WebserviceEvent"
```

In this example, the required fields are chosen within the XSL file, as the extra text required for the Netcool event format added manually.

Without the formatting the output would be on a single line, and not identify the field, only it's value.

```
<?xml version="1.0" encoding="UTF-8"?>xxxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxx1.01234567890WebserviceEvent
```

5.1.6 Example XML embedded event parsing

File : custom_embedded_event.xml

```
<notification>
  <alarm-notification>
    <alarm>
      <alarm-id>ID1234567890</alarm-id>
      <alarm-status>Alarming</alarm-status>
      <alarm-type-id>Alarm</alarm-type-id>
    </alarm>
  </alarm-notification>
</notification>
```

File : custom_embedded_event.xsl

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text" omit-xml-declaration="yes" />
<xsl:template match="/">
        <xsl:for-each select="notification/alarm-notification/alarm">
                <xsl:text>
alarm-id:string:"</xsl:text>
                <xsl:value-of select="alarm-id"/>
                <xsl:text>"
</xsl:text>
                <xsl:text>alarm-status:string:"</xsl:text>
                <xsl:value-of select="alarm-status"/>
                <xsl:text>"
</xsl:text>
                <xsl:text>alarm-type-id:string:"</xsl:text>
                <xsl:value-of select="alarm-type-id"/>
                <xsl:text>"
</xsl:text>

        </xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

```
./xslchecker.sh custom_embedded_event.xsl custom_embedded_event.xml

Output from applying transformer 'custom_embedded_event.xsl' to source file
'custom_embedded_event.xml':-

alarm-id:string:"ID1234567890"
alarm-status:string:"Alarming"
alarm-type-id:string:"Alarm"
```

5.2 JSON events

The probe and gateway use an internal JSON parser engine.

5.2.1 Simple JSON event

Refer to online references for acceptable JSON data syntax, such as www.w3schools.com.

JSON data must have the data quoted and embraced.

```
{"event_id":"123456789"}
```

Use industry standard JSON syntax checkers to confirm the JSON syntax of the incoming events before investigating why the probe did not process an event.

5.2.2 General JSON parsing method

The message bus probe uses the probe property settings to define the default JSON parsing, These can be used to determine the settings from a specific event source, before applying the settings to the JSON parser file, which allows different parsing for numerous endpoints.

Example JSON probe property settings.

```
MessagePayload    : 'json'
JsonNestedPayload : 'json'
JsonMessageDepth  : 10
TransformerFile   : ''
```

Example JSON parser file probe property settings.

```
MessagePayload   : 'json'
TransformerFile  : '${OMNIHOME}/probes/linux2x86/message_bus_prometheus_parser.json'
JsonMessageDepth : 10
```

5.2.3 Nested JSON storage example

The manual includes a number of JSON event parsing examples. These can be used to understand the way the probe parses events.

Example input data provided using Curl.

```
curl -v -H "Content-Type: application/json" -d '{"payload" :"{\"properties\":
{\"storage\": {\"type\": \"object\",\"oneOf\":
[{\"$ref\": \"#\/definitions\/diskDevice\"}, {\"$ref\":\"#\/definitions\/diskUUID\"},
{\"$ref\": \"#\/definitions\/nfs\"},
{\"$ref\":\"#\/definitions\/tmpfs\"}]},\"fstype\":{\"enum\":
[\"ext3\",\"ext4\",\"btrfs\"]},\"options\":
{\"type\":\"array\",\"minItems\":\"1\",\"items\":
{\"type\":\"string\"},\"uniqueItems\": \"true\"}}}", "header": {"options" :
"none"},"log":{"message":"Alert"}}' -X POST http://localhost:8080/
```

Example#1 probe property settings:

```
MessagePayload                  : 'json.payload'
JsonNestedPayload               : 'json'
```

Results:

```
properties.fstype.enum.0:  ext3
properties.fstype.enum.1:  ext4
properties.fstype.enum.2:  btrfs
properties.options.items.type:  string
properties.options.minItems:    1
properties.options.type:   array
properties.options.uniqueItems: true
properties.storage.oneOf.0.$ref:#/definitions/diskDevice
properties.storage.oneOf.1.$ref:#/definitions/diskUUID
properties.storage.oneOf.2.$ref:#/definitions/nfs
properties.storage.oneOf.3.$ref:#/definitions/tmpfs
properties.storage.type:   object
```

Example#2 probe property settings:

```
MessagePayload                  : 'json.payload'
JsonNestedPayload               : 'json.properties.storage'
```

Results:

```
oneOf.0.$ref:      #/definitions/diskDevice
oneOf.1.$ref:      #/definitions/diskUUID
oneOf.2.$ref:      #/definitions/nfs
oneOf.3.$ref:      #/definitions/tmpfs
resync_event:      false
type:      object
```

5.3 XML gateway settings

5.3.1 XML settings

The XML gateway allows for the use of the EmptyTransformer to allow the transformer modules XML engine to process events.

File : G_XML.props

```
Gate.XMLGateway.TransformerInputType  : 'xml'
Gate.XMLGateway.TransformerFile : '$NCHOME/omnibus/gates/G_XML/transformers.xml'
```

File : transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<tns:transformer name="netcoolEvents" type="northbound" endpoint="http://localhost:9090"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
</tns:transformer>
</tns:transformers>
```

5.3.2 JSON settings

The XML gateway allows for the use of the EmptyTransformer to allow the transformer modules JSON engine to process events.

File : G_JSON.props

```
Gate.XMLGateway.TransformerInputType  : 'json'
Gate.XMLGateway.TransformerFile : '$NCHOME/omnibus/gates/G_JSON/transformers.xml'
```

File : transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<tns:transformer name="netcoolEvents" type="northbound" endpoint="http://localhost:9090"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
</tns:transformer>
</tns:transformers>
```

The old method was to use a XML to JSON transformer.

File : G_JSON.props

```
Gate.XMLGateway.TransformerInputType  : 'xml'
Gate.XMLGateway.TransformerFile : '$NCHOME/omnibus/gates/G_JSON/transformers.xml'
```

File : transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<tns:transformer name="netcoolEvents" type="northbound" endpoint="http://localhost:9090"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
<tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2json.xsl" description="XSLT file for converting Netcool
events to json events"/>
</tns:transformer>
```

6 General Guidance

6.1 Custom Configurations

The Message bus probe and gateway packages install files in common directories. Because of this, it is best to create custom directory structures to hold copies of the edited configuration files.

Default file locations : Probe
$NCHOME/omnibus/probes/java/nco_p_message_bus.env
$NCHOME/omnibus/probes/j<platform>/nco_p_message_bus.props
$NCHOME/omnibus/probes/j<platform>/nco_p_message_bus.rules

Example Transport files:
Directory : $NCHOME/omnibus/java/conf
File extension : *.properties

XML XLS transformer files:
Directory : $NCHOME/omnibus/java/conf
File extension : *.xls

Example JSON parser files:
Directory : $NCHOME/omnibus/probes/j<platform>/
File extension : *.json

Default file locations : Gateway

Directory : $NCHOME/omnibus/gateway/xml
G_XML.props
xml.map
xml.reader.tblrep.def
xml.startup.cmd

default:
G_XML.props
xml.map
xml.reader.tblrep.def
xml.startup.cmd

Example Transport files:
Directory : $NCHOME/omnibus/java/conf
File extension : *.properties

XML XLS transformer files:
Directory : $NCHOME/omnibus/java/conf
File extension : *.xls

6.2 Kafka Server configuration

6.2.1 Using the groupi.d

The group.id is set in the probes kafkaClient.properties file, by default as:

```
group.id=KafkaConsumerGroupName
```

This can be used to nullify the need for the zookeeper server connection through the creation of a subscription group.

Listing the servers group.id's

```
bin/kafka-consumer-groups.sh  --list --bootstrap-server localhost:9092

KafkaConsumerGroupName
```

Adding the probes group.id to the Kafka server

```
bin/kafka-consumer-groups.sh --new-consumer --describe --group KafkaConsumerGroupName
--bootstrap-server localhost:9092
```

Now the zookeeper_client settings are no longer required in the kafkaConnectionProperties.json file.

6.2.2 Heartbeating topic

You can create a heartbeating topic which you can use to send period events to.
On the Kafka server with the correct java path set use the Kafka scripts to create a heartbeating topic.

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic heartbeating
```

Set the probes kafkaConnectionProperties.json file to reflect a subscription to the heartbeating topic.

```
 "topics": "topicABC,topicXYZ,heartbeating",
```

Start the message bus probe.

Try sending an message to the probe using the heartbeating topic.

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic heartbeating
>
{"eventfactory":[{"heartbeat-
id":"1234567890","timestamp":"date","summary":"Heartbeat","severity":"information"}]}
```

6.2.3 Example heartbeating script

```
#! /bin/sh
export KAFKADIR DATE PERIOD COUNT KJAVA
# set required paths and settings
KJAVA=/opt/jdk1.8.0_144/jre
KAFKADIR=/opt/kafka_2.12-1.1.0
PERIOD=60
# Initialise counter
COUNT=0
# Ensure the right Java is in the path
# as KAFKA is Java sensitive
PATH=${KJAVA}/bin:${PATH}

# Run forever
while true
do
# Count the heartbeats
COUNT=`expr $COUNT + 1`
# Create timestamp number
DATE=`date +'%Y%m%d%H%M%S'`
# Connect and send event
${KAFKADIR}/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic
heartbeating << EOF
{"eventfactory":[{"heartbeat-id":"$COUNT","timestamp":"$
{DATE}","summary":"Heartbeat","severity":"information"}]}
EOF
#  sleep for PERIOD seconds
sleep $PERIOD
done

#EOF
```

6.3 Lower KAFKA versions

The default third party transport JARs are provided with the Transport module package. It may be necessary to install the servers transport JARs and use these with the nco_p_message_bus probe or gateway.

For example, you may see these messages in the transport.log file:

```
DEBUG [Thread-20] internals.Fetcher (Fetcher.java:754) - [Consumer
clientId=BusProbe001, groupId=ProbeGroup] Skipping validation of fetch offsets for
partitions [heartbeating-0] since the broker does not support the required protocol
version (introduced in Kafka 2.3)
```

Which may cause issues when reading events from the KAFKA server.

You can either add the required JARs to the TransportModule.env file or if there are other KAFKA message bus applications using the file, update the file to pick up the environment settings, and use these instead of the default.

```
File : $NCHOME/omnibus/java/jars/TransportModule.env

# Allow custom KAFKA JARS
if [ -z "$APACHE_KAFKA_JARS" ]
then
APACHE_KAFKA_JARS=${TRANSPORT_JARS_DIR}/apache/kafka/kafka-clients-2.3.1.jar
else
echo "Using APACHE_KAFKA_JARS = $APACHE_KAFKA_JARS"
fi

if [ -z "$APACHE_ZOOKEEPER_JARS" ]
then
APACHE_ZOOKEEPER_JARS=${TRANSPORT_JARS_DIR}/apache/zookeeper/zookeeper-3.4.14.jar
else
echo "Using APACHE_ZOOKEEPER_JARS = $APACHE_ZOOKEEPER_JARS"
fi
```

The APACHE_KAFKA_JARS and APACHE_ZOOKEEPER_JARS environment variables can then set in the users environment:

```
# Set KAKFA JARS
APACHE_KAFKA_JARS=$NCHOME/omnibus/gates/G_KAFKA/java/kafka-clients-1.1.0.jar
APACHE_ZOOKEEPER_JARS=$NCHOME/omnibus/gates/G_KAFKA/java/zookeeper-3.4.10.jar
export APACHE_KAFKA_JARS APACHE_ZOOKEEPER_JARS
```

You can check they are picked up usig the -version option:

```
$NCHOME/omnibus/probes/nco_p_message_bus -version
Using APACHE_KAFKA_JARS = $NCHOME/omnibus/java/custom/kafka-clients-1.1.0.jar
Using APACHE_ZOOKEEPER_JARS = $NCHOME/omnibus/java/custom/zookeeper-3.4.10.jar
```

6.4 Higher KAFKA versions in probes

If the KAFKA server is using a higher version of KAFKA and the message bus probe is seeing unexpected issues, you can try to set the CLASSPATH and JRE to the same version as the KAFKA server. This usually resolves issues related to classes and Java mismatches.

File : $NCHOME/omnibus/probes/java/nco_p_message_bus.env

```
# KAFKA Java and JARs
# Adding KAFKA Server JARs to the CLASSPATH
MYJARPATH=$NCHOME/omnibus/probes/linux2x86/kafka/jars
export MYJARPATH
CLASSPATH=$MYJARPATH/kafka-log4j-appender-2.5.0.jar:$CLASSPATH
CLASSPATH=$MYJARPATH/kafka-clients-2.5.0.jar:$CLASSPATH
CLASSPATH=$MYJARPATH/zookeeper-3.5.7.jar:$CLASSPATH
echo "CLASSPATH=$CLASSPATH"

# Setting a Java
NCO_PROBE_JRE=/opt/sun-java-x86_64-80/jre
echo "NCO_PROBE_JRE=$NCO_PROBE_JRE"
```

The probes transport.log will include the KAFKA client version in the INFO logging.
e.g.

```
grep 'Kafka version:' transport.log
INFO  [ProbeRunner] utils.AppInfoParser$AppInfo (AppInfoParser.java:117) - Kafka version: 2.3.1
INFO  [ProbeRunner] utils.AppInfoParser$AppInfo (AppInfoParser.java:117) - Kafka version: 2.5.0
```

7 Message bus Probe Guidance

7.1 General guidance

To create a new probe integration it is best to create a custom directory and copy the required customisation files to this directory. The probe property file can remain in the default location. For JMS/MQTT integrations use the $NCHOME/omnibus/probes/java directory as the integration uses custom JAR files, whilst for other integrations use the $NCHOME/omnibus/probes/<platform> directory. Copy over the required configuration files and then edit them to suit the requirements of the integration. Afterwards, update the probes property file to suit the newly created directory and filenames.

Example HTTP JSON probe configuration

File locations:
$NCHOME/omnibus/probes/<platform>/message_bus_http_json.props
$NCHOME/omnibus/probes/<platform>/message_bus_http_json/httpTransport.properties
$OMNIHOME/probes/<platform>/message_bus_http_json/message_bus_parser_config.json

File creation:

```
cd $NCHOME/omnibus/probes/<platform>
cp message_bus.props  message_bus_http_json.props
mkdir  message_bus_http_json
cd  message_bus_http_json
cp $NCHOME/omnibus/java/conf/httpTransport.properties .
Vi  httpTransport.properties
serverPort=http:9191
:wq
cp $NCHOME/omnibus/probes/<platform>/message_bus_parser_config.json .
Cd cd $NCHOME/omnibus/probes/<platform>
Vi message_bus_http_json.props
Server          : 'AGG_P'
NetworkTimeout  : 15
PollServer      : 60
# HTTP/JSON
TransportType   : 'HTTP'
MessagePayload  : 'json'
TransportFile   : '$NCHOME/omnibus/probes/<platform>/message_bus_http_json/httpTransport.properties'
# JSON message parsing

JsonMessageDepth                : 10
# JsonNestedHeader                : ''
# JsonNestedPayload               : ''
# Default parsing
TransformerFile : ''
# Uncomment for custom parsing
# TransformerFile : '$OMNIHOME/probes/<platform>/message_bus_http_json/message_bus_parser_config.json'
#
HeartbeatInterval               : 0
#EOF
```

7.2 Creating a custom probe type

The Message bus probe is used for many integrations that are candidates for new probe integrations. It is possible to manually create these using the Message bus files, so that the probe can be easily integrated and managed.

For example Nokia NFPM integration could be manually turned into a new custom probe using the following steps.

```
cd $NCHOME/omnibus/probes
ln -s nco_jprobe nco_p_message_bus_nokia_nfmp
cd java
ln -s nco_p_message_bus.jar nco_p_message_bus_nokia_nfmp.jar
cp nco_p_message_bus.env nco_p_message_bus_nokia_nfmp.env
```

Required files and directories:

```
cd $NCHOME/omnibus/probes/linux2x86
mkdir  nco_p_message_bus_nokia_nfmp
mkdir  nco_p_message_bus_nokia_nfmp/JKS
mkdir  nco_p_message_bus_nokia_nfmp/default
```

JKS : A directory to hold the Java Store File

Copy the required configuration files to the default directory and message_bus_nokia_nfmp for modification:

message_bus_nokia_nfmp.props
message_bus_nokia_nfmp_parser.json
nokiaNspKafkaClient.properties
nokiaNspKafkaConnectionProperties.json
nokiaNspKafkaTransport.properties
nokiaNspRestMultiChannelHttpTransport.json

Note: If a new Message bus probe patch is installed, it is recommended that the new configuration files are copied to the default directory, as well as the probe jar and env files recreated, with the env file modified as required. Because the Message bus probe relies on the transformer and transport modules, any integration specific nco_p_message_bus.jar should be kept up to date.

7.3 Webhook transport payload

The Webhook transport property file accepts the undocumented feature, maxContentLength.

File : maxContentLengthWebhookTransport.properties

```
webhookURI=/
IdleTimeout=60
# Disable URL path checks
#respondWithContent=OFF
#validateBodySyntax=ON
validateRequestURI=OFF
# Force TLSv1.2
# securityProtocol=TLSv1.2
# Limit the accepted payload length
maxContentLength=20971520
# EOF
```

7.4 Proxy server configuration

The message bus probe can be configured for Proxy servers when using HTTP or HTTPS.

```
File : $NCHOME/omnibus/probes/java/nco_p_message_bus.env
```

```
###
# To enable PROXY server Host and Port for HTTP, HTTPS and SOCKS
# Set and uncomment the features that are required
# NCO_JPROBE_JAVA_FLAGS="-Dhttp.proxyHost=<PROXYIP> -Dhttp.proxyPort=<PROXYPORT>  ${NCO_JPROBE_JAVA_FLAGS}"
# NCO_JPROBE_JAVA_FLAGS="-Dhttps.proxyHost=<PROXYIP> -Dhttps.proxyPort=<PROXYPORT>  ${NCO_JPROBE_JAVA_FLAGS}"
# NCO_JPROBE_JAVA_FLAGS="-DsocksProxyHost=<PROXYIP> -DsocksProxyPort=<PROXYPORT> ${NCO_JPROBE_JAVA_FLAGS}"
# NCO_JPROBE_JAVA_FLAGS="-Djdk.http.auth.tunneling.disabledSchemes='' ${NCO_JPROBE_JAVA_FLAGS}"
```

Where <PROXYIP> and <PROXYPORT> are the proxy server IP address and port.
The SOCKS setting is included in case it is required.

8 XML Gateway Guidance

8.1 General guidance

To create a new gateway integration it is best to create a custom directory in the $NCHOME/omnibus/gates directory. This directory can then be populated with copies of the required configuration files before editing. For JMS/MQTT integrations add a jar file sub-directory. Copy over the required configuration files before making any modifications. Afterwards update the gateways property file to reflect the newly created directory and filenames.

Directory: $NCHOME/omnibus/gates/G_HTTP

Files:
G_HTTP.props
httpTransport.properties
transformers.xml
xml.map
xml.reader.tblrep.def
xml.startup.cmd

File creation:

```
cd $NCHOME/omnibus/gates
cp -r xml G_HTTP
cd G_HTTP
cp G_XML.props G_HTTP.props
cp  $NCHOME/omnibus/java/conf/httpTransport.properties .
Vi httpTransport.properties
clientURL=http://localhost:12345
#EOF
:wq
cp  $NCHOME/omnibus/java/conf/transformers.xml .
Vi transformers.xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <tns:transformer name="netcoolEvents" type="northbound" endpoint="http://localhost:12345"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
        </tns:transformer>
</tns:transformers>
:wq
vi xml.reader.tblrep.def
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'Poll = 1'
AFTER IDUC DO 'Poll = 2'
;
#EOF
:wq
```

8.2 Supporting packages

By default the probe and gateway packages installs the supporting packages when they are installed. However, if the supporting packages have been installed separately, they may not be updated when the latest probe or gateway package are installed.

To check the package listing use imcl:

~/IBM/InstallationManager/eclipse/tools/imcl listInstalledPackages -features -long | awk -F\: '{ print $3,$4}'

If only the probe and gateway package are listed then the latest supporting packages should have been installed as well.

If the supporting packages are listed as well, it is likely the latest supporting packages need to be installed separately.
e.g.

```
Netcool/OMNIbus Gateway nco-g-xml   1.9.0.0
Netcool/OMNIbus Common transformer   1.6.0.0
Netcool/OMNIbus Common transportmodule   1.11.0.0
```

To find the latest packages search the internet using the strings like:

```
ibm release transformer
ibm release transportmodule
```

And compare the release dates with the probe and gateway release date.

In order to apply the latest supporting package you will need to remove the old package before installing the new package.
e.g.

```
~/IBM/InstallationManager/eclipse/tools/imcl uninstall com.ibm.tivoli.omnibus.integrations.transportmodule
~/IBM/InstallationManager/eclipse/tools/imcl uninstall com.ibm.tivoli.omnibus.integrations.transformer

~/IBM/InstallationManager/eclipse/tools/imcl listAvailablePackages -repositories ./Im-common-transportmodule-18_0.zip
~/IBM/InstallationManager/eclipse/tools/imcl -c install com.ibm.tivoli.omnibus.integrations.transportmodule -repositories ./Im-common-transportmodule-18_0.zip
~/IBM/InstallationManager/eclipse/tools/imcl listAvailablePackages -repositories ./Im-common-transformer-8_0.zip
~/IBM/InstallationManager/eclipse/tools/imcl -c install com.ibm.tivoli.omnibus.integrations.transformer -repositories ./Im-common-transportmodule-18_0.zip
```

Check everything is installed as expected using imcl.

```
Netcool/OMNIbus Gateway nco-g-xml   1.9.0.0
Netcool/OMNIbus Common transformer   1.8.0.0
Netcool/OMNIbus Common transportmodule   1.18.0.0
```

Note: Install the latest fix pack for Netcool/OMNIbus as well to ensure all the supporting packages are up to date as libngtk is provided with the Netcool/OMNIbus fix pack.

8.2.1 nco_g_xml.env

There is a possibility that the nco_g_xml.env is not updated as expected. To check for this issue you can use UNIX find to locate the default env files, that are updated correctly, provided they remain unmodified.
e.g.

```
find $NCHOME -name nco_g_xml.env
/opt/nrv81/IBM/tivoli/netcool/omnibus/platform/linux2x86/bin/default/nco_g_xml.env
/opt/nrv81/IBM/tivoli/netcool/omnibus/platform/linux2x86/bin/nco_g_xml.env
/opt/nrv81/IBM/tivoli/netcool/omnibus/platform/linux2x86/bin64/default/nco_g_xml.env
/opt/nrv81/IBM/tivoli/netcool/omnibus/platform/linux2x86/bin64/nco_g_xml.env
```

The files can then be used to compare against the env files being used.
e.g.

```
diff $NCHOME/omnibus/platform/linux2x86/bin/default/nco_g_xml.env
$NCHOME/omnibus/platform/linux2x86/bin/nco_g_xml.env

diff $NCHOME/omnibus/platform/linux2x86/bin64/default/nco_g_xml.env
$NCHOME/omnibus/platform/linux2x86/bin64/nco_g_xml.env
```

If there are any unexpected differences try replacing the working env file with the default file, and add any debugging statements if required.
e.g.

```
cd $NCHOME/omnibus/platform/linux2x86/bin64
Ls -l nco_g_xml.env
mv nco_g_xml.env nco_g_xml.env.old
cp  default/nco_g_xml.env .
Ls -l nco_g_xml.env
```

8.3 Proxy configuration

The XML Gateways Gate.Java.Arguments property setting can be used to set the HTTP and HTTPS proxy settings.

File : G_HTTPS_JSON.props

```
Gate.Java.Arguments : "-Xmx4096m -Dhttp.proxyHost=<PROXYIP>
-Dhttp.proxyPort=<PROXYPORT> -Dhttps.proxyHost=<PROXYIP>
-Dhttps.proxyPort=<PROXYPORT> -Djdk.http.auth.tunneling.disabledSchemes='"
```

Where <PROXYIP> is the Proxy servers IP address and <PROXYPORT> is the Proxy servers port.
It's best to set both the HTTP and HTTPS Proxy server properties to ensure they are both set, in case one or other is required in a future configuration.

In the example the JVM memory limitation was already set to 4Gb.

8.4 Gateway performance Tuning

The XML Gateway may prove to be restricted in the volume of events it is able to forward. This is due to the single thread processing. The workaround for this is to use use multiple gateways, and filter the events usin the MOD function applied to the Serial or ServerSerial, as applicable.

For example, if five gateways are required, create five property and table replication files and modify them for use, setting the FILTER in the table replication uniquely for each gateway, so that that gateway only forwards one fifth of all events.

The example below gives the example property and table replication file for the first gateway, along with the table replication files for the other four gateways.

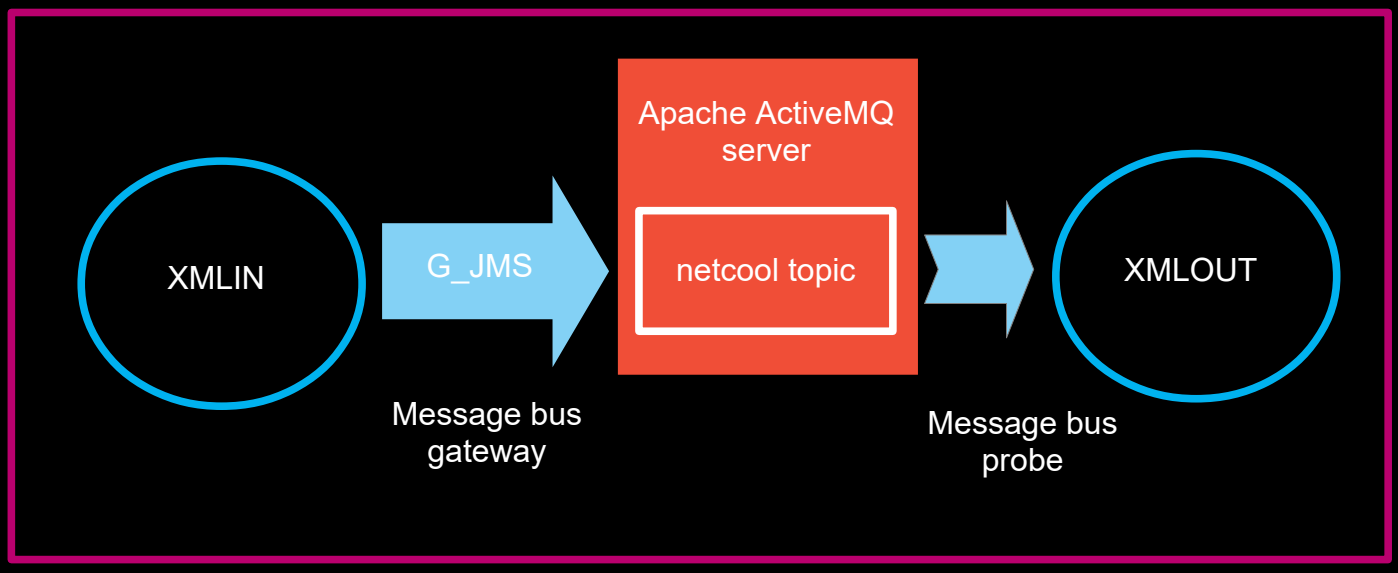
```
# Unique settings required
Name                            : 'G_HTTP01'
MessageLog                      : '$OMNIHOME/log/G_HTTP01.log'
Gate.Reader.TblReplicateDefFile : '$NCHOME/omnibus/gates/G_HTTP_Multiple/xml01.reader.tblrep.def'
# Common settings
Gate.XMLGateway.TransportType   : 'HTTP'
Gate.Reader.Server              : 'AGG_V'
MessageLevel                    : 'info'
# Files
Gate.XMLGateway.TransformerFile : '$NCHOME/omnibus/gates/G_HTTP_Multiple/transformers-json.xml'
Gate.XMLGateway.TransportFile   : '$NCHOME/omnibus/gates/G_HTTP_Multiple/httpTransport.properties'
Gate.MapFile                    : '$NCHOME/omnibus/gates/G_HTTP_Multiple/xml.map'
Gate.StartupCmdFile             : '$NCHOME/omnibus/gates/G_HTTP_Multiple/xml.startup.cmd'
MaxLogFileSize                  : 102400
Gate.XMLGateway.MessageKey      : ''
#EOF


File : $NCHOME/omnibus/gates/G_HTTP_Multiple/xml01.reader.tblrep.def

# Replicate MOD 5 Serial
REPLICATE INSERT, UPDATE  FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH '(mod(Serial,5)=0)'
;
# EOF

File : $NCHOME/omnibus/gates/G_HTTP_Multiple/xml02.reader.tblrep.def

# Replicate MOD 5 Serial
REPLICATE INSERT, UPDATE  FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH '(mod(Serial,5)=1)'
;
# EOF
File : $NCHOME/omnibus/gates/G_HTTP_Multiple/xml03.reader.tblrep.def

# Replicate MOD 5 Serial
REPLICATE INSERT, UPDATE  FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH '(mod(Serial,5)=2)'
;
# EOF
File : $NCHOME/omnibus/gates/G_HTTP_Multiple/xml04.reader.tblrep.def

# Replicate MOD 5 Serial
REPLICATE INSERT, UPDATE  FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH '(mod(Serial,5)=3)'
;
# EOF
File : $NCHOME/omnibus/gates/G_HTTP_Multiple/xml05.reader.tblrep.def

# Replicate MOD 5 Serial
REPLICATE INSERT, UPDATE  FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH '(mod(Serial,5)=4)'
;
# EOF
```

Probe and Gateway Examples

The following chapters cover how to use the probe and gateway together which is useful for testing transports and transformer configurations

9 Example Apache ActiveMQ Probe and Gateway usage

The following design was used to configure a probe and gateway to forward events from one object server to another via an Apache ActiveMQ sever, to illustrate message bus usage and configuration. This type of configuration is not practical but does provide a useful test environment for users to familiarise themselves with the technologies used by the message bus integration.



9.1 The Message Bus JMS Gateway

9.1.1 G_JMS.props

This is an example property file:

```
Name                                 : 'G_JMS'
Gate.Reader.Server                   : 'XMLIN'
Ipc.Timeout                          : 600
Gate.Reader.IducFlushRate            : 11
Gate.XMLGateway.TransportType        : 'JMS'
MessageLog                           : '$OMNIHOME/log/G_JMS.log'
PropsFile                            : '$OMNIHOME/gates/G_JMS/G_JMS.props'
Gate.MapFile                         : '$OMNIHOME/gates/G_JMS/xml.map'
Gate.StartupCmdFile                  : '$OMNIHOME/gates/G_JMS/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile      : '$OMNIHOME/gates/G_JMS/xml.reader.tblrep.def'
Gate.XMLGateway.TransformerFile      : '$OMNIHOME/gates/G_JMS/conf/transformers.xml'
Gate.XMLGateway.TransportFile        : '$OMNIHOME/gates/G_JMS/conf/jmsTransport.properties'
Gate.XMLGateway.MessageID            : 'netcoolEvents'
Gate.Java.ClassPath                  :
'/opt/IBM/tivoli/netcool/omnibus/gates/java/nco_g_xml.jar:/opt/IBM/tivoli/netcool/omnibus/gates/java/ngjava.jar:
/opt/IBM/tivoli/netcool/omnibus/java/jars/TransportModule.jar:/opt/IBM/tivoli/netcool/omnibus/java/jars/Transfor
mer.jar:/opt/IBM/tivoli/netcool/omnibus/java/jars/com.ibm.ws.ejb.thinclient_8.5.0.jar:/opt/IBM/tivoli/netcool/om
nibus/java/jars/com.ibm.ws.sib.client.thin.jms_8.5.0.jar:/opt/IBM/tivoli/netcool/omnibus/java/jars/JSON4J.jar:/o
pt/IBM/tivoli/netcool/omnibus/gates/G_JMS/conf/slf4j-api-
1.7.13.jar:/opt/IBM/tivoli/netcool/omnibus/gates/G_JMS/conf/log4j-
1.2.17.jar:/opt/IBM/tivoli/netcool/omnibus/gates/G_JMS/conf/activemq-all-
5.13.0.jar:/opt/IBM/tivoli/netcool/omnibus/gates/G_JMS/conf'
```

9.1.2 xml.reader.tblrep.def

The table replication only needs to forward the alerts.status table, with the SENT_TO_AAMQ flag added as a filter and set to 1 after the events have been forwarded to the Apache ActiveMQ server:

```
REPLICATE ALL FROM TABLE 'alerts.status'
        USING MAP 'StatusMap'
FILTER WITH 'SENT_TO_AAMQ=0'
AFTER IDUC DO 'SENT_TO_AAMQ=1'
;
```

9.1.3 Adding a control flag

Add a control flag to the source object server, XMLIN, so that the events can be marked as sent:

```
alter table alerts.status add SENT_TO_AAMQ int;
go
```

9.1.4 Creating the G_JMS directory

In order to create a new XML Gateway configuration it is best to create this in the $OMNIHOME/gates directory, adding all the required configuration files, so that they are kept in one place and therefore are not modified whilst deploying another Message Bus integration.

```
cd $OMNIHOME/gates
cp -r xml G_JMS
cd G_JMS
mkdir conf
```

Copy the required jar files to the conf directory and update or create the required configuration files.

Once the configuration is complete the directory will contain the following files:

```
G_JMS:
G_JMS.props
conf
xml.map
xml.reader.tblrep.def
xml.startup.cmd

G_JMS/conf:
transformers.xml
jmsTransport.properties
jndi.properties
log4j-1.2.17.jar
slf4j-api-1.7.13.jar
activemq-all-5.13.0.jar
```

Example settings:

9.1.5 transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- Northbound (gateway) transformer definitions -->
        <tns:transformer name="netcoolEvents" type="northbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
        </tns:transformer>
</tns:transformers>
```

9.1.6 jmsTransport.properties

```
initialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory
providerURL=tcp://192.168.20.20:61616
topicName=netcool
topicConnectionFactory=ConnectionFactory
#queueName=netcool
#queueConnectionFactory=ConnectionFactory
username=activemq
password=activemq
#EOF
```

9.1.7 jndi.properties

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url = vm://192.168.20.20
connectionFactoryNames = ConnectionFactory
#queue.netcool = netcool
topic.netcool = netcool
#EOF
```

9.2 The Message Bus JMS probe

The message bus probes configuration requires the same sort of configuration, so you can use some of the files, like jndi.properties and the jmsTransport.properties from the gateways configuration.

Create a probe configuration directory in the $OMNIHOME/probes/java directory, and copy or create the configuration files, then copy the jar files there, so that the directory has the following files:

```
transformers.xml
jmsTransport.properties
jndi.properties
activemq-all-5.13.0.jar
log4j-1.2.17.jar
slf4j-api-1.7.13.jar
```

9.2.1 message_bus.props

The message bus property file defines what the transport type is and where to find the transport properties and transformers definition file:

```
Server            : 'XMLOUT'
#ServerBackup     : ''
NetworkTimeout    : 15
PollServer        : 30
# JMS Transport
TransportType     : 'JMS'
TransportFile     : '/opt/IBM/tivoli/netcool/omnibus/probes/java/nco_p_message_bus/jmsTransport.properties'
# Transformer
MessagePayload    : 'xml'
TransformerFile   : '/opt/IBM/tivoli/netcool/omnibus/probes/java/nco_p_message_bus/transformers.xml'
WebSocketID       : 'netcoolEvents'
#EOF
```

9.2.2 nco_p_message_bus.env

Add the jar file paths to the probe environment script as shown:

```
#####################################################
# Add on required jar files for Apache ActiveMQ
#####################################################
export jarfile
CLASSPATH=${OMNIHOME}/probes/java/${PROGRAM}:$CLASSPATH
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/log4j-1.2.17.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/slf4j-api-1.7.13.jar
# Uncomment for the MQTT transport
#CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/wmqtt.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/activemq-all-5.13.0.jar
# DEBUGGING
echo "CLASSPATH= " $CLASSPATH
echo $CLASSPATH | awk -F\: '{for(i=1;i<=NF;i++)print $i}' | while read jarfile
do
echo $jarfile
done
#EOF
```

9.2.3 transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Southbound (probe) transformer defintions -->
<tns:transformer name="netcool2nvpairs" type="southbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
<tns:property name="xsltFilename" type="java.lang.String"
value="$OMNIHOME/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting
Netcool events to name/value pairs"/>
</tns:transformer>
</tns:transformers>
```

9.2.4 jmsTransport.properties

```
initialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory
providerURL=tcp://192.169.20.20:61616
#queueName=netcool
#queueConnectionFactory=ConnectionFactory
topicName=netcool
topicConnectionFactory=ConnectionFactory
username=activemq
password=activemq
#EOF
```

9.2.5 jndi.properties

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url = vm://192.168.20.20
connectionFactoryNames = ConnectionFactory
#queue.netcool = netcool
topic.netcool = netcool
#EOF
```

9.3 Event processing

9.3.1 Creating an SQL user

In order to check the object servers process the test events it is best to create a didicated user for use with the SQL scripts. Add this user to both XMLIN and XMLOUT:

```
create user 'eventuser' id 555 full name 'eventuser' password 'eventuser';
go
alter group 'Probe' assign members  'eventuser';
go
alter group 'ISQL' assign members  'eventuser';
go
alter group 'ISQLWrite' assign members  'eventuser';
go
```

9.3.2 Sending test events

To test event processing and event needs to be inserted into the source object server, XMLIN. This can be done using the following script:

File : insert_event.sh

```
#! /bin/sh
SQLCMD="$OMNIHOME/bin/nco_sql -server XMLIN -user eventuser -password eventuser"
export SQLCMD DATE
DATE=`date`
echo $DATE
$SQLCMD <<EOF
insert into alerts.status
(Identifier,Severity,Summary,AlertGroup,AlertKey,OwnerUID,OwnerGID)
values ('test-alert-${DATE}',4,'XML Gateway test event','testing','Gateway
event',65534,0);
go
quit
EOF
#EOF
```

9.3.3 Checking event processing

The inserted events can be checked to see if they have been processed, which will occur when SENT_TO_AAMQ is set to 1 by the Message Bus gateway:

File : check_GW_event.sh

```
#! /bin/sh
SQLCMD="$OMNIHOME/bin/nco_sql -server XMLIN -user eventuser -password eventuser"
export SQLCMD DATE
$SQLCMD <<EOF
select count(*) from alerts.status where AlertGroup = 'testing' and SENT_TO_AAMQ = 1;
go
quit
EOF
#EOF
```

To check the target object server, XMLOUT, you can use the following script:

File : check_event.sh

```
#! /bin/sh
SQLCMD="$OMNIHOME/bin/nco_sql -server XMLOUT -user eventuser -password eventuser"
export SQLCMD DATE
$SQLCMD <<EOF
select count(*) from alerts.status where Summary = 'XML Gateway test event' ;
go
quit
EOF
#EOF
```

9.3.4 The Results

To test the behaviour, login to the Apache ActiveMQ server using a browser and the admin/admin user. Monitor the clients connecting and the number of events processed. Start the G_JMS gateway, and then the probe in debug mode. Using tail to monitor the log files is useful, but not necessary unless an issue is seen.

Insert ten events into the XMLIN object server using the insert_event.sh script.
Monitor the SENT_TO_AAMQ flag using check_GW_event.sh script, and once the flag is set to 1, check for the event in the XMLOUT object server using the check_event.sh script.

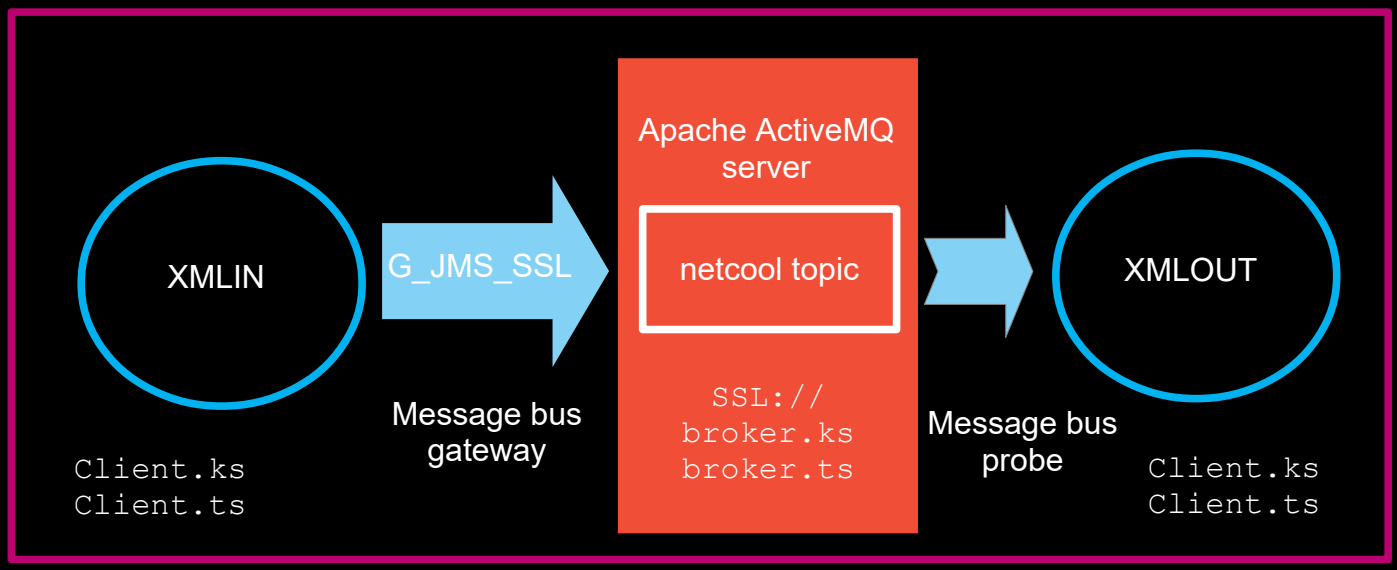
With ten unique events, all ten events should be seen in the target object server within a minute, having the original Summary, if the default rules files are being used.

If necessary, check the probe is receiving events using the Apache ActiveMQ web page. Click the 'send' tab and enter the Topic name 'netcool' and select Topic, from the menu. Copy and paste the test event given in General Guidance chapter, into the 'Message Body' and click 'Send'. The probes log should indicate that the event was tokenised and processed, with this event being visible in the XMLOUT's event list.

General comment:
The MTTrapd probe was used to perform a simple throughput test. It was noted that the event rate between the source probe, creating the events, and the Message Bus probe were of the same order. Although it was noted that there were periodic pauses in the Message Bus probes event processing that may indicate a possible bottleneck in its event processing. For example, if the MTTrapd probe ran at 20 events per second, the Message Bus probe could run at the same event rate, but its event rate would periodically reduce to 5 events per second for one 10s log interval. It is advised the customers check the Message Bus integrations performance within their own environment before deploying a solution to ensure that it is capable of managing the load it will be required to handle.

One workaround for any performance issues would be to run multiple probes and/or gateways and uses filters to divide the load amongst them.

10Example Apache ActiveMQ Probe and Gateway with SSL usage



10.1 Apache ActiveMQ Server

The example installation was performed on Linux using the apache-activemq-5.13.0 package.

The Apache ActiveMQ Server uses four JKS files:

1. broker.ks - Apache ActiveMQ server keystore
2. broker.ts – Apache ActiveMQ server truststore
3. client.ks – client keystore
4. client.ts -client truststore

The TLS/SSL certificates can be selfsigned certificates or Certificate Authority certificates.
The Apache ActiveMQ sets the location of JKS files to the conf directory and defines their location using the servers 'env' file. Once configured the activemq.xml can be switched to ssl.

Users environment:
Set the directories for the Netcool/OMNIbus probe java, and Apache ActiveMQ.
This is not necessary, but reduces the risk of Java security issues.

```
NCHOME /opt/nrv81/IBM/tivoli/netcool
JAVA_HOME $NCHOME/platform/linux2x86/jre64_1.8.0/jre
ACTIVEMQ_HOME /opt/apache-activemq-5.13.0
export NCHOME JAVA_HOME ACTIVEMQ_HOME
```

File : bin/env

From:

```
ACTIVEMQ_SSL_OPTS=""
```

To:

```
# Enable SSL Java keystores
ACTIVEMQ_SSL_OPTS="-Djavax.net.ssl.keyStore=$ACTIVEMQ_HOME/conf/broker.ks
-Djavax.net.ssl.keyStorePassword=netcool
-Djavax.net.ssl.trustStore=$ACTIVEMQ_HOME/conf/broker.ts
-Djavax.net.ssl.trustStorePassword=netcool"
```

File : conf/activemq.xml

From:

```
<transportConnector name="openwire" uri="tcp://0.0.0.0:61616?
maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
```

To:

```
 <transportConnector name="openwire" uri="ssl://0.0.0.0:61616?
maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
```

Directory : conf
Install the new self-signed certificates in the Apache ActiveMQ servers conf directory and check the details.

```
keytool -list  -keystore conf/broker.ks  -storepass netcool
keytool -list  -keystore conf/broker.ts  -storepass netcool
keytool -list  -keystore conf/client.ks  -storepass netcool
keytool -list  -keystore conf/client.ts  -storepass netcool
```

10.1.1 Simple SSL script

The Apache ActiveMQ manual includes steps to create self-signed certificates for test purposes. The following script can be used to create the self-signed certificates, and the required JKS files used by example. You must preserve the newlines when creating the script, and respond to command prompts [netcool|yes] when the script is run. The script will create the files in the current working directory. The default host is localhost for both probe and Apache ActiveMQ server. The use of an host as an alias is part of the newer SSL security requirements.

File : create_broker_client_certs.sh

```
#! /bin/sh
keytool -genkey -alias localhost -keyalg RSA -keystore broker.ks -storepass netcool <<EOF
localhost




US
yes

EOF
ls -l broker.ks

keytool -export -alias localhost -keystore broker.ks -file broker_cert
ls -l broker_cert

keytool -genkey -alias localhost -keyalg RSA -keystore client.ks -storepass netcool <<EOF
localhost




US
yes

EOF
ls -l client.ks

keytool -import -alias localhost -keystore client.ts -file broker_cert

keytool -export -alias localhost -keystore client.ks -file client_cert
ls -l client_cert

keytool -import -alias localhost -keystore broker.ts -file client_cert

echo "Checking : "
keytool -printcert -file client_cert
keytool -printcert -file broker_cert
keytool -list  -keystore broker.ks  -storepass netcool
keytool -list  -keystore broker.ts  -storepass netcool
keytool -list  -keystore client.ks  -storepass netcool
keytool -list  -keystore client.ts  -storepass netcool

# EOF
```

10.2 ActiveMQ Probe with SSL

10.2.1 Overview

The following files were used to configure the message bus probe.

Directory: $NCHOME/omnibus/probes/linux2x86/JMS_SSL:
```
message_bus.props
jmsTransport.properties
```

Directory: $NCHOME/omnibus/probes/linux2x86/JMS_SSL/JARS:
```
activemq-all-5.13.0.jar
log4j-1.2.17.jar
slf4j-log4j12-1.7.13.jar
```

Directory: $NCHOME/omnibus/probes/linux2x86/JMS_SSL/SSL:
```
broker.ks - for reference
broker.ts - for reference
client.ks
client.ts
```

10.2.2 Environment settings

These settings can be used in the probe uses environment, or added to the probes environment file.

Required JARs for Apache ActiveMQ
```
CLASSES=$NCHOME/omnibus/probes/linux2x86/JMS_SSL/JARS
export CLASSES
CLASSPATH=${CLASSES}:${CLASSES}/activemq-all-5.13.0.jar:${CLASSES}/log4j-1.2.17.jar:$
{CLASSES}/slf4j-log4j12-1.7.13.jar
export CLASSPATH
```

Required trust store settings, passed to the probe using the NCO_JPROBE_JAVA_FLAGS setting:
```
CERTS=${NCHOME}/omnibus/probes/linux2x86/JMS_SSL/SSL
export CERTS
NCO_JPROBE_JAVA_FLAGS="-Djavax.net.ssl.trustStore=${CERTS}/client.ts
-Djavax.net.ssl.trustStorePassword=netcool $NCO_JPROBE_JAVA_FLAGS"
export NCO_JPROBE_JAVA_FLAGS
```

Example CSHRC file

```
setenv CLASSES $NCHOME/omnibus/probes/linux2x86/JMS_SSL/JARS
setenv CLASSPATH ${CLASSES}:${CLASSES}/activemq-all-5.13.0.jar:${CLASSES}/log4j-1.2.17.jar:${CLASSES}/slf4j-
log4j12-1.7.13.jar

setenv CERTS ${NCHOME}/omnibus/probes/linux2x86/JMS_SSL/SSL
setenv NCO_JPROBE_JAVA_FLAGS "-Djavax.net.ssl.trustStore=${CERTS}/client.ts
-Djavax.net.ssl.trustStorePassword=netcool"
```

10.2.3 Property file

File : message_bus.props

```
Name                 : "message_bus_apache_active_mq_ssl"
RulesFile            : "$NCHOME/omnibus/probes/linux2x86/message_bus.rules"
# Debug
# MessageLevel         : 'debug'
# Production
MessageLevel         : 'info'
# Object Server
Server               : 'AGG_P'
ServerBackup         : 'AGG_B'
# Best practice settings
NetworkTimeout       : 15
PollServer           : 60
# Buffering
Buffering            : 1
BufferSize           : 200
FlushBufferInterval  : 11
# Tuning
DisableDetails       : 1
MaxEventQueueSize    : 50000
# JMS Transport settings
TransportType        : 'JMS'
TransportFile        : '$NCHOME/omnibus/probes/linux2x86/JMS_SSL/jmsTransport.properties'
# Default JSON event parsing
TransformerFile      : ''
MessagePayload       : 'json'
JsonMessageDepth     : 10
# Disable heartbeating
HeartbeatInterval    : 0
# EOF
```

File jmsTransport.properties

```
initialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory
providerURL=ssl://<FQDN>:61616
topicConnectionFactory=ConnectionFactory
queueConnectionFactory=ConnectionFactory
topicName=netcool
username=admin
password=admin
```

10.2.4 Checking the probe connection

The probe can be started from the command line:

```
cd $NCHOME/omnibus/probes/JMS_SSL
$NCHOME/omnibus/probes/nco_p_message_bus -propsfile ./message_bus.props -messagelevel debug -messagelog stdout
```

This will create a connection to the Apache ActiveMQ server which visible in the servers administration console in the topics tab [http://localhost:8161/admin/]. Clicking on the 'Send To' link for the probes connection displays the event form.
Enter an example JSON event, click 'send' and check the probes debug log messages for event.

```
{"message":"testing 1 2 3"}
```

Note: Use the Apache ActiveMQ servers administration console for checking the XML gateways connection too.

10.3 ActiveMQ Gateway with SSL

10.3.1 Overview

The XML Gateway is configured for JMS using the Apache ActiveMQ jars. The SSL configuration of the client keystore and truststore is defined in the gateways property file. In this example the events are sent in the JSON format. The XML gateway requires the jndi.properties file to be configured and a single JMS topic.

Directory: $NCHOME/omnibus/gates/G_JMS_SSL

```
G_JMS_SSL.props
xml.map
xml.reader.tblrep.def
xml.startup.cmd

./SSL:
broker.ks
broker.ts
client.ks
client.ts

./conf:
jmsTransport.properties
jndi.properties
log4j.properties
transformers.xml

./java:
activemq-all-5.13.0.jar
log4j-1.2.17.jar
slf4j-log4j12-1.7.13.jar
```

10.3.2 Configuration

The JMS Gateway requires the Apache ActiveMQ server JAR files as well as the client and broker keystore and truststore files.

Directory: $NCHOME/omnibus/gates/G_JMS_SSL/java

```
activemq-all-5.13.0.jar
log4j-1.2.17.jar
slf4j-log4j12-1.7.13.jar
```

The CLASSPATH can be set in the gateways user environment, or the gateways environment file.

```
CLASSPATH=$OMNIHOME/gates/G_JMS_SSL/conf
CLASSPATH=$CLASSPATH:$OMNIHOME/gates/G_JMS_SSL/conf/log4j.properties
CLASSPATH=$CLASSPATH:$OMNIHOME/gates/G_JMS_SSL/java/activemq-all-5.13.0.jar
CLASSPATH=$CLASSPATH:$OMNIHOME/gates/G_JMS_SSL/java/log4j-1.2.17.jar
export CLASSPATH
```

Directory: $NCHOME/omnibus/gates/G_JMS_SSLSSL

```
broker.ks
broker.ts
client.ks
client.ts
```

Directory: $NCHOME/omnibus/gates/G_JMS_SSL/conf

```
jmsTransport.properties
jndi.properties
log4j.properties
transformers.xml
```

File : jmsTransport.properties

```
initialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory
# JMS settings
# TCP - can be used to test connectivity
#providerURL=tcp://<FQDN>:61616
# SSL
providerURL=ssl://<FQDN>:61616
# Only define one destination
topicConnectionFactory=ConnectionFactory
topicName=netcool
# Default admin user name and password
username=admin
password=admin
#EOF
```

File : jndi.properties

```
java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url=vm://<FQDN>
connectionFactoryNames=ConnectionFactory
topic.netcool=netcool
#EOF
```

File : transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Northbound (gateway) transformer definitions -->
<tns:transformer name="netcoolNVP" type="northbound" endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
</tns:transformer>
</tns:transformers>
```

Directory: $NCHOME/omnibus/gates

File : G_JMS_SSL.props

```
Name                                  : 'G_JMS_SSL'
Gate.Reader.Server                    : 'AGG_P'
# Set file names
MessageLog                            : '$OMNIHOME/log/G_JMS_SSL.log'
PropsFile                             : '$OMNIHOME/gates/G_JMS_SSL/G_JMS_SSL.props'
Gate.MapFile                          : '$OMNIHOME/gates/G_JMS_SSL/xml.map'
Gate.StartupCmdFile                   : '$OMNIHOME/gates/G_JMS_SSL/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile       : '$OMNIHOME/gates/G_JMS_SSL/xml.reader.tblrep.def'
# JMS configuration
Gate.XMLGateway.TransportType         : 'JMS'
Gate.XMLGateway.TransportFile         : '$OMNIHOME/gates/G_JMS_SSL/conf/jmsTransport.properties'
# JSON configuration
Gate.XMLGateway.TransformerInputType  : 'json'
Gate.XMLGateway.MessageID             : 'netcoolNVP'
Gate.XMLGateway.TransformerFile       : '$OMNIHOME/gates/G_JMS_SSL/conf/transformers.xml'
Gate.Reader.IducFlushRate             : 11
MessageLevel                          : 'info'
# Debugging
# MessageLevel                          : 'debug'
# Gate.Reader.LogOSSql                  : TRUE

# SSL keystore configuration
Gate.Java.Arguments                   : "-Djavax.net.ssl.keyStore=$OMNIHOME/gates/G_JMS_SSL/SSL/client.ks
-Djavax.net.ssl.keyStorePassword=netcool -Djavax.net.ssl.trustStore=$OMNIHOME/gates/G_JMS_SSL/SSL/client.ts
-Djavax.net.ssl.trustStorePassword=netcool"
```

File : xml.reader.tblrep.def

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'Poll=1'
AFTER IDUC DO 'Poll=2'
;
```

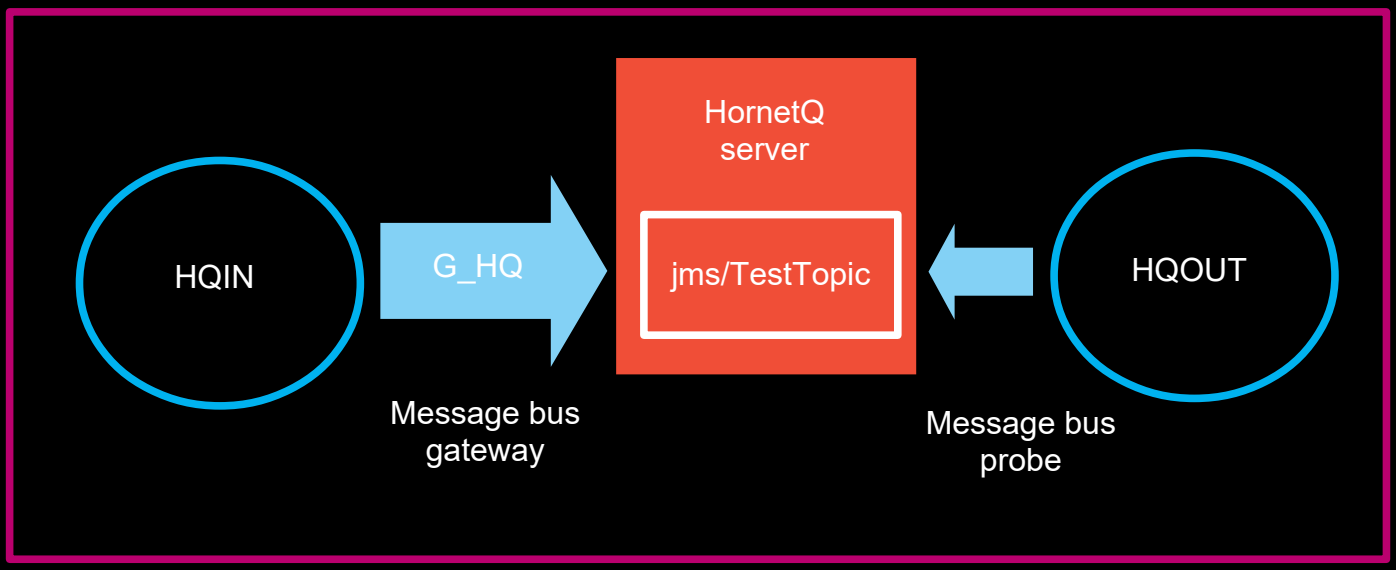
File : xml.startup.cmd

```
<empty file>
```

File : xml.map

```
CREATE MAPPING StatusMap
(
        'Identifier'        =    '@Identifier'           ON INSERT ONLY,
        'Node'              =    '@Node'                 ON INSERT ONLY,
        'NodeAlias'         =    '@NodeAlias'            ON INSERT ONLY
                                                                NOTNULL '@Node',
        'Manager'           =    '@Manager'              ON INSERT ONLY,
        'Agent'             =    '@Agent'                ON INSERT ONLY,
        'AlertGroup'        =    '@AlertGroup'           ON INSERT ONLY,
        'AlertKey'          =    '@AlertKey'             ON INSERT ONLY,
        'Severity'          =    '@Severity',
        'Summary'           =    '@Summary',
        'StateChange'       =    '@StateChange',
        'FirstOccurrence'   =    '@FirstOccurrence'      ON INSERT ONLY,
        'LastOccurrence'    =    '@LastOccurrence',
        'InternalLast'      =    '@InternalLast',
        'Poll'              =    '@Poll'                 ON INSERT ONLY,
        'Type'              =    '@Type'                 ON INSERT ONLY,
        'Tally'             =    '@Tally',
        'Class'             =    '@Class'                ON INSERT ONLY,
        'Grade'             =    '@Grade'                ON INSERT ONLY,
        'Location'          =    '@Location'             ON INSERT ONLY,
        'OwnerUID'          =    '@OwnerUID',
        'OwnerGID'          =    '@OwnerGID',
        'Acknowledged'      =    '@Acknowledged',
        'Flash'             =    '@Flash',
        'EventId'           =    '@EventId'              ON INSERT ONLY,
        'ExpireTime'        =    '@ExpireTime'           ON INSERT ONLY,
        'ProcessReq'        =    '@ProcessReq',
        'SuppressEscl'      =    '@SuppressEscl',
        'Customer'          =    '@Customer'             ON INSERT ONLY,
        'Service'           =    '@Service'              ON INSERT ONLY,
        'ServerName'        =    '@ServerName'           ON INSERT ONLY,
        'ServerSerial'      =    '@ServerSerial'         ON INSERT ONLY
);
```

11 Example HornetQ probe and gateway configuration



The HornetQ server is a lightweight messaging server capable of transmitting JMS messages. The following configuration uses the default installation and server configuration. Both the probe and gateway can be configured to connect to the same topic and pass events between themselves. For this to work there must be two object servers, HQIN and HQOUT.

The HornetQ server uses localhost by default, so additional configuration is required if the probe and gateway need to be remote for testing.

11.1 Configuring HornetQ

The HornetQ server can be downloaded from the website http://hornetq.jboss.org. It is unpacked into the local directory, and is ready for use immediately for the given examples.

11.1.1 Creating a Test Topic

The TestTopic is created in the hornetq-jms.xml file.
Default HornetQ configuration files are all located in *INSTALL_DIR*/config/stand-alone/non-clustered.

To add a new Connection Factory "TestTopicConnectionFactory" and Topic "TestTopic",
add the following connection-factory clause at the end of the connection-factory settings:

```
<connection-factory name="TestTopicConnectionFactory" signature="topic">
   <xa>false</xa>
   <connectors>
      <connector-ref connector-name="netty"/>
   </connectors>
   <entries>
      <entry name="jms/TestTopicConnectionFactory"/>
   </entries>
</connection-factory>
```

And after the defaults queues add:

```
<topic name="TestTopic">
   <entry name="jms/TestTopic"/>
</topic>
```

To allow the HornetQ server to be connected to remotely replace the localhost with the HornetQ servers IP Address:

hornetq-beans.xml:

```
<property name="bindAddress">${jnp.host:<HORNETQ_IP>}</property>
<property name="rmiBindAddress">${jnp.host:<HORNETQ_IP>}</property>
```

hornetq-configuration.xml:

```
<param key="host"  value="${hornetq.remoting.netty.host:<HORNETQ_IP>}"/>
<param key="host"  value="${hornetq.remoting.netty.host:<HORNETQ_IP>}"/>
<param key="host"  value="${hornetq.remoting.netty.host:<HORNETQ_IP>}"/>
<param key="host"  value="${hornetq.remoting.netty.host:<HORNETQ_IP>}"/>
```

Set the debug logging in the logging.properties file to check connectivity:

```
cd config/stand-alone/non-clustered
vi  logging.properties
logger.level=DEBUG
```

and set the full path to the logs directory:

```
handler.FILE.fileName=/opt/hornetq-2.4.0.Final/logs/hornetq.log
```

The HornetQ is started by executing run.sh in the bin directory.
e.g.

```
cd /opt/hornetq-2.4.0.Final/bin
./run.sh
```

11.2 JMS probe

The following details provide a working example for the HornetQ server.

Directory: $NCHOME/omnibus/probes/java/nco_p_message_bus

Contents:

- hornetq-commons.jar
- hornetq-core-client.jar
- hornetq-jms-client.jar
- jboss-jms-api.jar
- jnp-client.jar
- jmsTransport.properties
- transformers.xml

11.2.1 transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">

        <!-- Southbound (probe) transformer defintions -->
        <tns:transformer name="netcool2nvpairs" type="southbound" endpoint="jms/TestTopic"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
                <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting Netcool events to name/value
pairs"/>
        </tns:transformer>

</tns:transformers>
```

11.2.2 jmsTransport.properties

```
initialContextFactory=org.jnp.interfaces.NamingContextFactory
providerURL=jnp://<HORNETQ_IP>:1099
topicConnectionFactory=jms/TestTopicConnectionFactory
topicName=jms/TestTopic
username=guest
password=guest
#EOF
```

11.2.3 nco_p_message_bus.env

Directory: $NCHOME/omnibus/probes/java/nco_p_message_bus

```
################################################################################
# Add on required jar files for HornetQ
################################################################################
CLASSPATH=${OMNIHOME}/probes/java/${PROGRAM}:$CLASSPATH
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/hornetq-core-client.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/hornetq-commons.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/hornetq-jms-client.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/jboss-jms-api.jar
CLASSPATH=$CLASSPATH:${OMNIHOME}/probes/java/${PROGRAM}/jnp-client.jar
echo "CLASSPATH= " $CLASSPATH
echo $CLASSPATH | awk -F\: '{for(i=1;i<=NF;i++)print $i}'  | while read jarfile
do
echo $jarfile
done
#EOF
```

11.2.4 message_bus_hornetq.props

Directory: $NCHOME/omnibus/probes/<platform>

```
Server : 'HQOUT'
TransportType   : 'JMS'
MessagePayload  : 'xml'
TransformerFile : '$NCHOME/omnibus/probes/java/nco_p_message_bus/transformers.xml'
TransportFile   : '$NCHOME/omnibus/probes/java/nco_p_message_bus/jmsTransport.properties'
#EOF
```

11.3 JMS Gateway

Directory : $NCHOME/omnibus/gates/G_HQ

Contents:

- CSHRC
- G_HQ.props
- conf
- xml.map (default)
- xml.reader.tblrep.def (default)
- xml.startup.cmd (default)

File : CSHRC

```
setenv CLASSPATH $OMNIHOME/gates/G_HQ/conf/hornetq-core-client.jar
setenv CLASSPATH $OMNIHOME/gates/G_HQ/conf/hornetq-commons.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_HQ/conf/hornetq-jms-client.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_HQ/conf/jboss-jms-api.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_HQ/conf/jnp-client.jar:$CLASSPATH
setenv CLASSPATH $OMNIHOME/gates/G_HQ/conf/netty.jar:$CLASSPATH

#EOF
```

11.3.1 G_HQ.props

```
Name : 'G_HQ'
MessageLog : '$OMNIHOME/log/G_HQ.log'
PropsFile : '$OMNIHOME/gates/G_HQ/G_HQ.props'
Gate.MapFile : '$OMNIHOME/gates/G_HQ/xml.map'
Gate.StartupCmdFile : '$OMNIHOME/gates/G_HQ/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile : '$OMNIHOME/gates/G_HQ/xml.reader.tblrep.def'
Gate.XMLGateway.TransformerFile : '$OMNIHOME/gates/G_HQ/conf/transformers.xml'
Gate.XMLGateway.TransportFile : '$OMNIHOME/gates/G_HQ/conf/jmsTransport.properties'
Gate.XMLGateway.TransportType : 'JMS'
Gate.XMLGateway.MessageID : 'netcoolEvents'
```

11.3.2 Required JAR files

Directory : $NCHOME/omnibus/gates/G_HQ/conf

Contents:

- hornetq-commons.jar
- hornetq-core-client.jar
- hornetq-jms-client.jar
- jboss-jms-api.jar
- jnp-client.jar
- netty.jar
- jmsTransport.properties
- transformers.xml

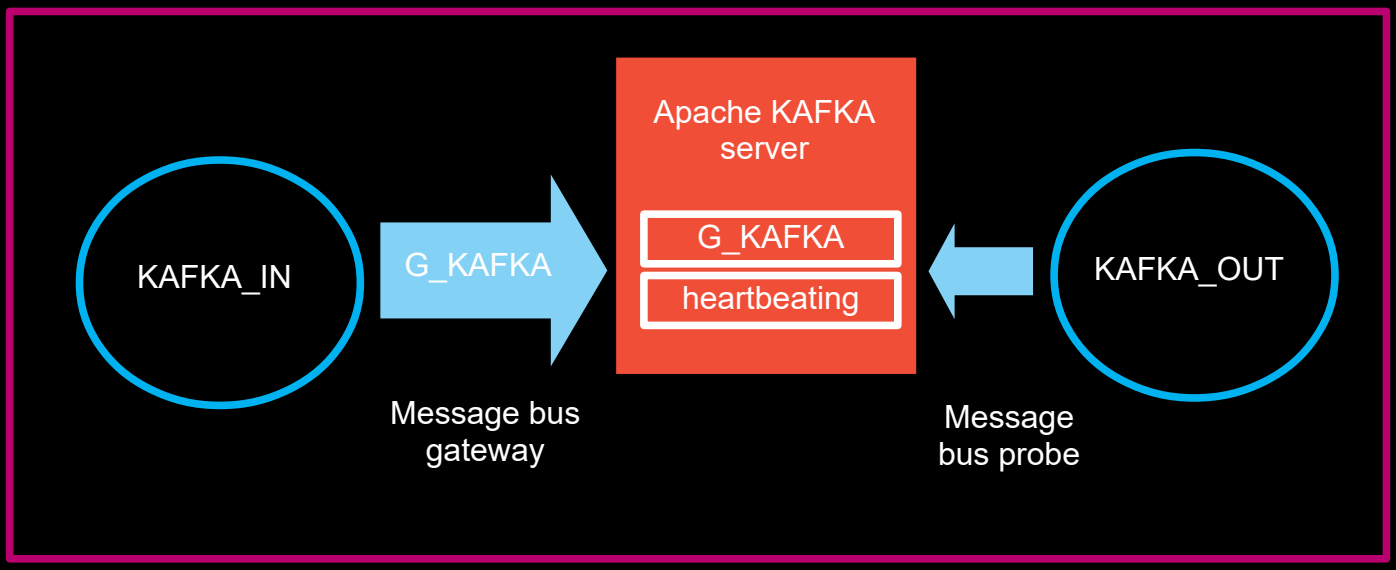
11.3.3 transformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Northbound (gateway) transformer definitions -->
<tns:transformer name="netcoolEvents" type="northbound" endpoint="jms/TestTopic"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
</tns:transformer>
</tns:transformers>
```

11.3.4 jmsTransport.properties

```
initialContextFactory=org.jnp.interfaces.NamingContextFactory
providerURL=jnp://<HORNETQ_IP>:1099
topicConnectionFactory=jms/TestTopicConnectionFactory
topicName=jms/TestTopic
username=guest
password=guest
#EOF
```

12Example Apache KAFKA Probe and Gateway Configuration



As with the Message bus integrations it is possible to feed the message bus probe using the message bus [XML] gateway. These types of integration examples are useful in understanding the Netcool/OMNIbus features and how the probe and gateway are configured. There is no attempt to performance tune or understand the impact of feeding the probe using the gateway, on the Apache KAFKA server.

In order to feed events from the gateway to the probe, the gateway [producer] needs to sends JSON event messages to the probes [consumer] topic.

12.1 Apache KAFKA Configuration

The software can be downloaded as a GNU compressed tar file which can be extracted into a directory.

Download : https://kafka.apache.org

Once unpacked the Apache KAFKA server can be started using the following commands:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
```

To prepare the server create the topics:

- heatbeating
- G_KAFKA

Using the commands:

```
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1
--topic heartbeating
bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1
--topic G_KAFKA
```

To check the topics are available use in zookeeper:

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

List consumer groups:

```
bin/kafka-consumer-groups.sh  --list --bootstrap-server localhost:9092
```

12.2 Gateway Configuration

This example uses the $NCHOME/omnibus/gates/G_KAFKA directory as the configuration directory. Using a single directory makes checking the files easier and prevents them from being overwritten by other integrations and package updates. The G_KAFKA server needs to be added to the XML Gateway servers $NCHOME/etc/omni.dat file.

12.2.1 G_KAFKA.props

The XML Gateways property file is the top level configuration file.
In this example xml.startup.cmd is the default file or an empty file.
The xml.map file can be the default map file or some customised mapping file, and is used to define the fields that are passed to KAFKA for each event.
The transformers.xml file needs to convert the Object Server events to JSON messages for consumption by the KAFKA server.

```
MessageLog                              : '$OMNIHOME/log/G_KAFKA.log'
Name                                    : 'G_KAFKA'
Gate.Reader.Server                      : 'KAFKA_IN'
MessageLevel                            : 'debug'
Gate.XMLGateway.TransportType           : 'KAFKA'
# Gate.XMLGateway.MessageID             : 'netcoolEvents'
Gate.XMLGateway.TransformerFile         : '$OMNIHOME/gates/G_KAFKA/transformers.xml'
Gate.XMLGateway.TransportFile           : '$OMNIHOME/gates/G_KAFKA/kafkaTransport.properties'
Gate.Reader.TblReplicateDefFile         : '$OMNIHOME/gates/G_KAFKA/xml.reader.tblrep.def'
Gate.MapFile                            : '$OMNIHOME/gates/G_KAFKA/xml.map'
Gate.StartupCmdFile                     : '$OMNIHOME/gates/G_KAFKA/xml.startup.cmd'
# EOF
```

12.2.2 kafkaTransport.properties

The main transport properties file defines the transport connection properties file.

```
KafkaClientMode=PRODUCER
ConnectionPropertiesFile=/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/G_KAFKA/kafkaConnectionProperties.json
#EOF
```

12.2.3 kafkaConnectionProperties.json

The KAFKA transport connection properties file defines the servers, transports, topics and properties files used for connecting to the KAFKA server. The latest version of KAFKA supports configurations without the need to define the zookeeper server, which simplifies the configuration. In this example the data is sent to the G_KAFKA topic in PLAINTEXT and without any authentication.

```
{
    "zookeeper_client" :
        {
            "target" : "",
            "properties" : "",
            "java_sys_props" : "",
            "topic_watch": false,
            "broker_watch": false
        },
    "brokers" : "PLAINTEXT://<FQDN>:9092",
    "topics": "G_KAFKA",
    "kafka_client" :
        {
            "properties" : "/opt/nrv81/IBM/tivoli/netcool/omnibus/gates/G_KAFKA/kafkaClient.properties",
            "java_sys_props" : ""
        }
}
```

12.2.4 kafkaClient.properties

The default KAFKA properties usually suffice, with the client.id being set to a unique string for identification within the KAFKA server.

```
client.id=KafkaNetcoolProducer
acks=all
max.block.ms=60000
retries=0
batch.size=16384
buffer.memory=33554432
```

12.2.5 transformers.xml

The transformers.xml file needs to be configured to redirect the netcoolEvents to the JSON XSLT parser, so that the Object Server events are passed to the KAFKA server in the correct format, JSON, and not the default XML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <tns:transformer name="netcoolEvents" type="northbound"  endpoint="netcool"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
                <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2json.xsl" description="XSLT file for converting Netcool events to JSON events"
/>
        </tns:transformer>
</tns:transformers>
```

12.2.6 xml.reader.tblrep.def

In this example the events passed to KAFKA are limited using a FILTER, so that only events with the Poll field set to a value are passed to the KAFKA server.

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'Poll>0'
;
```

12.3 Probe Configuration

The Message bus probe is configured like the KAFKA XML gateway using the same sort of properties.
It is best to place all the configuration files in one direction to ensure they are not modified by another integration or a package installation. In this example the events are forwarded to a new object server to allow them to be checked for successful transmission.

12.3.1 message_bus_kafka.G_KAFKA.props

You can set the Name property to a unique name if more than one Message bus probe is being used on the same server, to ensure the filenames are unique. In this example only the pertinent properties are set. The Transformer file used is the default JSON transformer file, and the rules file is a custom rules file to allow suitable event processing and event logging.

```
Server                      : 'KAFKA_OUT'
NetworkTimeout              : 15
PollServer                  : 60
MessageLevel                : 'debug'
Manager                     : 'Kafka'
MessageLog                  : '$OMNIHOME/log/message_bus_G_KAFKA.log'
MaxLogFileSize              : 10485760
RulesFile                   : '$OMNIHOME/probes/linux2x86/message_bus_kafka.rules'
TransportType               : 'KAFKA'
TransportFile               : '$OMNIHOME/probes/linux2x86/G_KAFKA/kafkaTransport.properties'
TransformerFile             : '$OMNIHOME/probes/linux2x86/G_KAFKA/message_bus_parser_config.json'
MessagePayload              : 'JSON'
#EOF
```

12.3.2 kafkaTransport.properties

The KAFKA properties defines the probe as a CONSUMER and sets the KAFKA connection properties file.

```
kafkaClientMode=CONSUMER
connectionPropertiesFile=$OMNIHOME/probes/linux2x86/G_KAFKA/kafkaConnectionProperties.json.nozoo
#EOF
```

12.3.3 kafkaConnectionProperties.json.nozoo

In this example the zookeeper server is not configured.
Only the KAFKA server and topics are defined along with the KAFKA client properties.

```
{
    "zookeeper_client" :
        {
            "target" : "",
            "properties" : "",
            "java_sys_props" : "",
            "topic_watch": true,
            "broker_watch": true
        },
    "brokers" : "PLAINTEXT://<FQDN>:9092",
    "topics": "G_KAFKA,heartbeating",
    "kafka_client" :
        {
            "properties" :
"/opt/nrv81/IBM/tivoli/netcool/omnibus/probes/linux2x86/G_KAFKA/kafkaClient.properties",
            "java_sys_props" : ""
        }
}
```

12.3.4 kafkaClient.properties

The KAFKA properties requires unique group.id and client.id settings. You could configure other probes with the same group.id to share event processing. For Peer-to-Peer configuration the group.id would need to be unique. The default serialization uses LongSerializer, which is incompatible with the events produced by the XML gateway.

```
group.id=GKAFKAMessageBusProbeGroup
pollInterval=1000
max.poll.records=1000
enable.auto.commit=true
auto.commit.interval.ms=5000
auto.offset.reset=latest
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
client.id=GKAFKAMessageBusProbe
acks=all
max.block.ms=60000
retries=0
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer
#EOF
```

12.4 Example Scripts

12.4.1 Heartbeat Script

The heartbeat.sh script is run on the KAFKA server from the KAFKA installation directory with the correct user environment. The script can be used to create a test event on the heartbeating topic, or to periodically send an event to check the probes connectivity.

File : heartbeat.sh

```
#! /bin/sh
export DATE
DATE=`date +%Y%m%d%H%M%S`
echo $DATE
echo '{"eventfactory":[{"heartbeat-id":"$
{DATE}","timestamp":"date","summary":"Heartbeat","severity":"information"}]}'
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic heartbeating <<
EOF
{"eventfactory":[{"heartbeat-id":"$
{DATE}","timestamp":"date","summary":"Heartbeat","severity":"information"}]}

EOF
#EOF
```

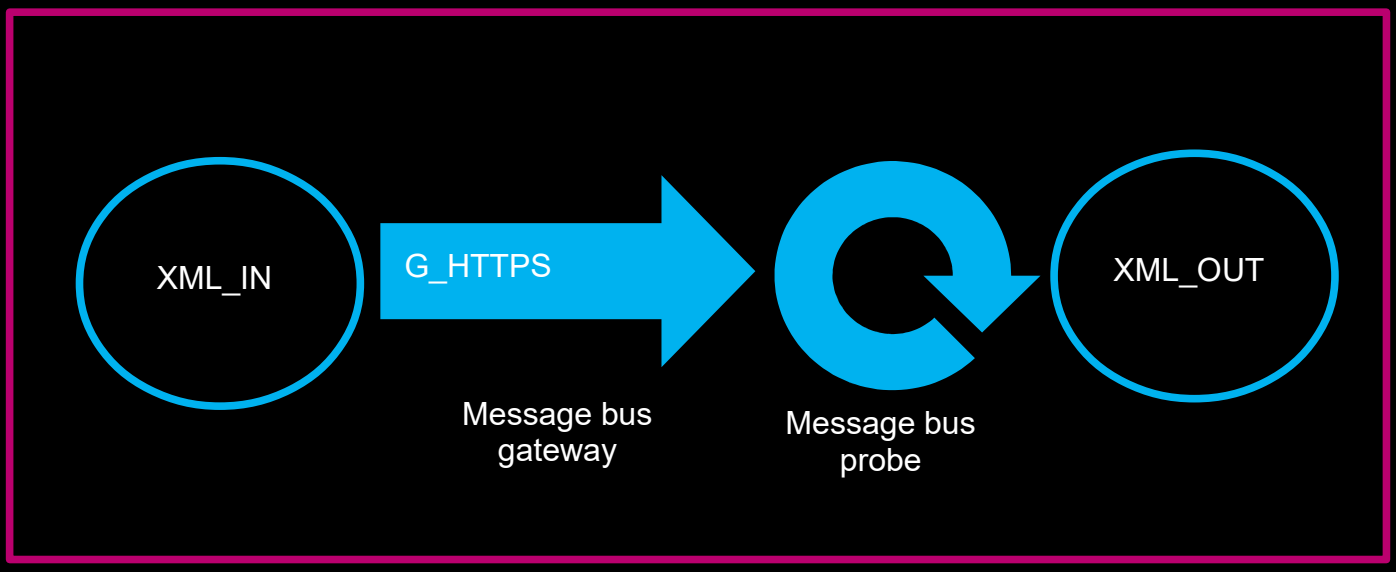
12.4.2 Test event script

The test event can be used to insert an event into the KAFKA_IN object server such that the event is passed to the KAFKA server by the G_KAFKA gateway. The correct username and password need to use to allow nco_sql to be used.

File : insert_poll_event.sh

```
#! /bin/sh
if [ $# -ne 1 ]
then
echo "Usage : `basename $0` NCOMS"
exit
fi


SQLCMD="$OMNIHOME/bin/nco_sql -server ${1} -user root -password netcool"
export SQLCMD DATE COUNTER DATE
DATE=`date`

$SQLCMD <<EOF
insert into alerts.status
(Identifier,Severity,Summary,AlertGroup,AlertKey,OwnerUID,OwnerGID,Poll)
values  ('test-poll-alert-example-${DATE}',4,
'Poll TEST event ${DATE}','testing','Poll',65534,0,1);
go
quit
EOF
#EOF
```

13Example Secure HTTP Transport with XML messages



The Message bus Gateway can send secure XML messages to the Message bus probe using HTTPS. In this example configuration a local CA is used to provide the CA certificates and sign the probes client certificate, so as to mimic production environments. Before implementing HTTPS, the example configuration will illustrate how to test the system using HTTP, and how to debug SSL communications.

The example configuration is for Red Hat Linux where platform is set to linux2x86.

13.1 HTTP probe

Create a basic HTTP probe configuration that is capable of reading XML Name Value Pairs.

13.1.1 message_bus_xml.props

```
Server          : 'XML_IN'
MessageLevel    : 'debug'
TransportType   : 'HTTP'
MessagePayload  : 'xml'
TransformerFile : '$NCHOME/omnibus/probes/linux2x86/messagebus-xml/transformer.xml'
TransportFile   : '$NCHOME/omnibus/probes/linux2x86/messagebus-
xml/httpsTransport.properties'
HeartbeatInterval : 60
```

13.1.2 httpsTransport.properties

Directory : $NCHOME/omnibus/probes/linux2x86/messagebus-xml

```
serverPort=http:5555
```

13.1.3 transformer.xml

Directory : $NCHOME/omnibus/probes/linux2x86/messagebus-xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Southbound (probe) transformer defintions -->
        <tns:transformer name="netcool2nvpairs" type="southbound" endpoint="http:5555"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
        <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting Netcool
events to name/value pairs"/>
        </tns:transformer>
</tns:transformers>
```

13.2 HTTP Gateway

Configure The G_HTTPS gateway to use HTTP initially.
Remember to add G_HTTPS as a server in $NCHOME/etc/omni.dat.

13.2.1 G_HTTPS.props

Directory : $OMNIHOME/gates/G_HTTPS/G_HTTPS.props

```
# SSL Debugging
# Gate.Java.Arguments : '-Djavax.net.debug=ssl:handshake:verbose'
#
Gate.XMLGateway.TransportType : 'HTTP'
Gate.Reader.Server : 'XML_IN'
Name : 'G_HTTPS'
MessageLevel : 'debug'
MessageLog : '$OMNIHOME/log/G_HTTPS.log'
PropsFile : '$OMNIHOME/gates/G_HTTPS/G_HTTPS.props'
Gate.MapFile : '$OMNIHOME/gates/G_HTTPS/xml.map'
Gate.StartupCmdFile : '$OMNIHOME/gates/G_HTTPS/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile : '$OMNIHOME/gates/G_HTTPS/xml.reader.tblrep.def'
Gate.XMLGateway.TransformerFile : '$OMNIHOME/gates/G_HTTPS/httpstransformers.xml'
Gate.XMLGateway.TransportFile : '$OMNIHOME/gates/G_HTTPS/httpsTransport.properties'
Gate.XMLGateway.MessageID : 'netcoolEvents'
#EOF
```

13.2.2 httpsTransport.properties

```
clientURL=http://<FQDN>:5555
bufferSize=1
#EOF
```

13.2.3 httpstransformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Northbound (gateway) transformer definitions -->
<tns:transformer name="netcoolEvents" type="northbound" endpoint="http://<FQDN>:5555"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
</tns:transformer>
</tns:transformers>
```

13.2.4 xml.reader.tblrep.def

```
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'Poll=1'
AFTER IDUC DO 'Poll=2'
;
#EOF
```

13.2.5 xml.startup.cmd

<empty>

13.2.6 xml.map

```
CREATE MAPPING StatusMap
(
        'Identifier'        =    '@Identifier'          ON INSERT ONLY,
        'Node'              =    '@Node'                ON INSERT ONLY,
        'NodeAlias'         =    '@NodeAlias'           ON INSERT ONLY
                                                        NOTNULL '@Node',
        'Manager'           =    '@Manager'             ON INSERT ONLY,
        'AlertGroup'        =    '@AlertGroup'          ON INSERT ONLY,
        'AlertKey'          =    '@AlertKey'            ON INSERT ONLY,
        'Severity'          =    '@Severity',
        'Summary'           =    '@Summary',
        'FirstOccurrence'   =    '@FirstOccurrence'     ON INSERT ONLY,
        'LastOccurrence'    =    '@LastOccurrence',
        'Type'              =    '@Type'                ON INSERT ONLY,
        'Tally'             =    '@Tally',
        'ServerName'        =    '@ServerName'          ON INSERT ONLY,
        'ServerSerial'      =    '@ServerSerial'        ON INSERT ONLY
);
```

13.3 Testing the event processing

Object Server User

```
create user 'testuser' id 888 full name 'Test User' password 'netcool';
go
alter group 'Probe' assign members  'testuser' ;
go
alter group 'ISQL' assign members  'testuser' ;
go
```

Insert Script

```
vi insert_GHTTPS_event.sh

#! /bin/sh
# Usage check
if [ $# -ne 1 ]
then
echo "Usage : `basename $0` NCOMS"
exit
fi
# SQL Command
SQLCMD="$OMNIHOME/bin/nco_sql -server ${1} -user testuser -password netcool"
export SQLCMD DATE COUNTER DATE
DATE=`date`
# Run command - Poll=1
$SQLCMD <<EOF
insert into alerts.status
(Identifier,Severity,Summary,AlertGroup,AlertKey,OwnerUID,OwnerGID,Poll)
values  ('test-alert-4-G_HTTPS-${DATE}',4,'TEST event
G_HTTPS','testing','G_HTTPS',65534,0,1);
go
quit
EOF

#EOF

chmod 755 insert_GHTTPS_event.sh
```

Insert an event into the object server [XML_IN]

Start the probe and gateway in debug mode and send an event to the probe using the gateway by adding an event to the source object server XML_IN using the script:

```
./insert_GHTTPS_event.sh XML_IN
```

The event should be seen in the gateways log file, and then after a time in the probes log file where it should be converted to an event, depending upon how the rules file was configured. This event should then be inserted into the target object server, XML_OUT.

13.4 Adding the SSL Certificates

Note : The probe requires the <FQDN> to be used for the signed SSL certificate.

Setting the correct Java environment

```
cd $NCHOME
find . -name keytool
./IBM/tivoli/netcool/platform/linux2x86/jre64_1.7.0/jre/bin/keytool

csh
set path=($NCHOME/platform/linux2x86/jre64_1.7.0/jre/bin $path)
rehash
java -version
```

13.4.1 Configuring the probe

Creating the probe keystore

```
keytool -genkey -alias <FQDN> -keystore ProbeStore.jks -keyalg RSA -sigalg
SHA1withRSA -storepass netcool
keytool -export -alias <FQDN> -file probe.cer -keystore ProbeStore.jks -storepass
netcool
```

Creating a local CA

```
setenv RANDFILE rand
openssl req -new -keyout cakey.pem -out careq.pem
openssl x509 -signkey cakey.pem -req -days 3650 -in careq.pem -out caroot.cer
-extensions v3_ca
keytool -printcert -v -file caroot.cer
```

Signing the probes certificate

```
keytool -certreq -alias <FQDN> -keystore ProbeStore.jks -file ProbeReq.csr

echo 1234>serial.txt
openssl x509 -CA caroot.cer -CAkey cakey.pem -CAserial serial.txt -req -in
ProbeReq.csr -out ProbeCA.cer -days 365
```

Loading the signed certificate into the probes keystore

```
keytool -import -keystore ProbeStore.jks -alias rootca -file caroot.cer
-storepass netcool
keytool -import -keystore ProbeStore.jks -alias <FQDN> -file ProbeCA.cer
-storepass netcool


keytool -list -keystore ProbeStore.jks -storepass netcool
Your keystore contains 2 entries
rootca, Jul 20, 2018, trustedCertEntry,
<FQDN>, Jul 20, 2018, keyEntry,
```

Setting the probes Transport file

Update 'http' to 'https'.
Add the JKS files and passwords.

File : httpsTransport.properties

```
serverPort=https:5555
trustStore=/tmp/ProbeStore.jks
trustStorePassword=netcool
keyStore=/tmp/ProbeStore.jks
keyStorePassword=netcool
```

Setting the probes Transformer file
Update 'http' to 'https'.

File : transformer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <!-- Southbound (probe) transformer defintions -->
        <tns:transformer name="netcool2nvpairs" type="southbound"
endpoint="https:5555"
className="com.ibm.tivoli.netcool.integrations.transformer.XSLTTransformer">
        <tns:property name="xsltFilename" type="java.lang.String" value="$
{OMNIHOME}/java/conf/netcool2nvpairs.xsl" description="XSLT file for converting
Netcool events to name/value pairs"/>
        </tns:transformer>
</tns:transformers>
```

Testing the probe using curl

Use the caroot.cer certificate file as the CA CERTIFICATE.

```
curl --cacert ./caroot.cer -v -H "Content-Type: application/json" -d
'{"test":"Message text"}' -X POST https://<FQDN>:5555
```

13.4.2 Configuring the Gateway

Create the Gateways keystore

```
keytool -import -keystore GatewayStore.jks -alias rootca -file ./caroot.cer
-storepass netcool
keytool -list -keystore GatewayStore.jks -storepass netcool
```

Gateways property file settings

File : G_HTTPS.props

```
# Uncomment for SSL debugging through stdout
#Gate.Java.Arguments : '-Djavax.net.debug=ssl:handshake:verbose'
#
Gate.XMLGateway.TransportType : 'HTTP'
Gate.Reader.Server : 'XML_IN'
Name : 'G_HTTPS'
MessageLog : '$OMNIHOME/log/G_HTTPS.log'
PropsFile : '$OMNIHOME/gates/G_HTTPS/G_HTTPS.props'
Gate.MapFile : '$OMNIHOME/gates/G_HTTPS/xml.map'
Gate.StartupCmdFile : '$OMNIHOME/gates/G_HTTPS/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile : '$OMNIHOME/gates/G_HTTPS/xml.reader.tblrep.def'
Gate.XMLGateway.TransformerFile : '$OMNIHOME/gates/G_HTTPS/httpstransformers.xml'
Gate.XMLGateway.TransportFile :
'$OMNIHOME/gates/G_HTTPS/httpsTransport.properties'
Gate.XMLGateway.MessageID : 'netcoolEvents'
#EOF
```

File : httpsTransport.properties

```
clientURL=https://<FQDN>:5555
trustStore=/tmp/GatewayStore.jks
keyStore=/tmp/GatewayStore.jks
trustStorePassword=netcool
keyStorePassword=netcool
#EOF
```

File : httpstransformers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Northbound (gateway) transformer definitions -->
<tns:transformer name="netcoolEvents" type="northbound"
endpoint="https://<FQDN>:5555"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
</tns:transformer>
</tns:transformers>
```

13.4.3 Example certificates

File : ProbeStore.jks

```
Keystore type: jks
Keystore provider: IBMJCE

Your keystore contains 2 entries

Alias name: rootca
Entry type: trustedCertEntry
Owner: EMAILADDRESS=root@<FQDN>, CN=<FQDN>, OU=Support, O=IBM, L=New York, ST=New
York, C=US
Issuer: EMAILADDRESS=root@<FQDN>, CN=<FQDN>, OU=Support, O=IBM, L=New York,
ST=New York, C=US

Alias name: <FQDN>
Owner: CN=<FQDN>, OU=Support, O=IBM, L=New York, ST=New York, C=US
Issuer: EMAILADDRESS=root@<FQDN>, CN=<FQDN>, OU=Support, O=IBM, L=New York,
ST=New York, C=US
```

File : GatewayStore.jks

```
Keystore type: jks
Keystore provider: IBMJCE
Your keystore contains 1 entry
Alias name: rootca
Entry type: trustedCertEntry
Owner: EMAILADDRESS=root@<FQDN>, CN=<FQDN>, OU=Support, O=IBM, L=New York, ST=New
York, C=US
Issuer: EMAILADDRESS=root@<FQDN>, CN=<FQDN>, OU=Support, O=IBM, L=New York,
ST=New York, C=US
```

Using the property file to set keystore files:

```
Gate.Java.Arguments : '-Djavax.net.ssl.keyStore=/tmp/GatewayStore.jks
-Djavax.net.ssl.keyStorePassword=netcool
-Djavax.net.ssl.trustStore=/tmp/GatewayStore.jks
-Djavax.net.ssl.trustStorePassword=netcool'
```

13.5 Debugging SSL

13.5.1 Probe Debugging

The Probe SSL messages can be debugged using the nco_p_message_bus.env file.

File : nco_p_message_bus.env
Directory : $NCHOME/omnibus/probes/java

```
NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=ssl:handshake:verbose"
echo "NCO_JPROBE_JAVA_FLAGS=$NCO_JPROBE_JAVA_FLAGS"
```

Running the probe:
Directory : $NCHOME/omnibus/probes/<platform>

```
$NCHOME/omnibus/probes/nco_p_message_bus -propsfile ./message_bus.props >
/tmp/message_bus_probe.ssl.log
```

Reading the log file:

```
grep -i found  message_bus_probe.ssl.log
found key for : <FQDN>

grep '\*\*\*' message_bus_probe.ssl.log
*** ClientHello, TLSv1.2
*** ServerHello, TLSv1.2
*** Certificate chain
*** ECDH ServerKeyExchange
*** ServerHelloDone
*** ECDHClientKeyExchange
*** Finished
*** Finished
*** ClientHello, TLSv1.2
*** ServerHello, TLSv1.2
*** Finished
*** Finished
*** ClientHello, TLSv1.2
*** ServerHello, TLSv1.2
*** Finished
*** Finished
```

13.5.2 Gateway Debugging

The Gateways SSL messages can be debugged using the Gate.Java.Arguments property. The details is written to standard out so will need to be redirected to a file. Remember to disable SSL logging after the issue is resolved.

File : G_HTTPS.props

```
# Uncomment for SSL debugging through stdout
Gate.Java.Arguments : '-Djavax.net.debug=ssl:handshake:verbose'
```

Running the Gateway:
Directory : $NCHOME/omnibus/gates/G_HTTPS

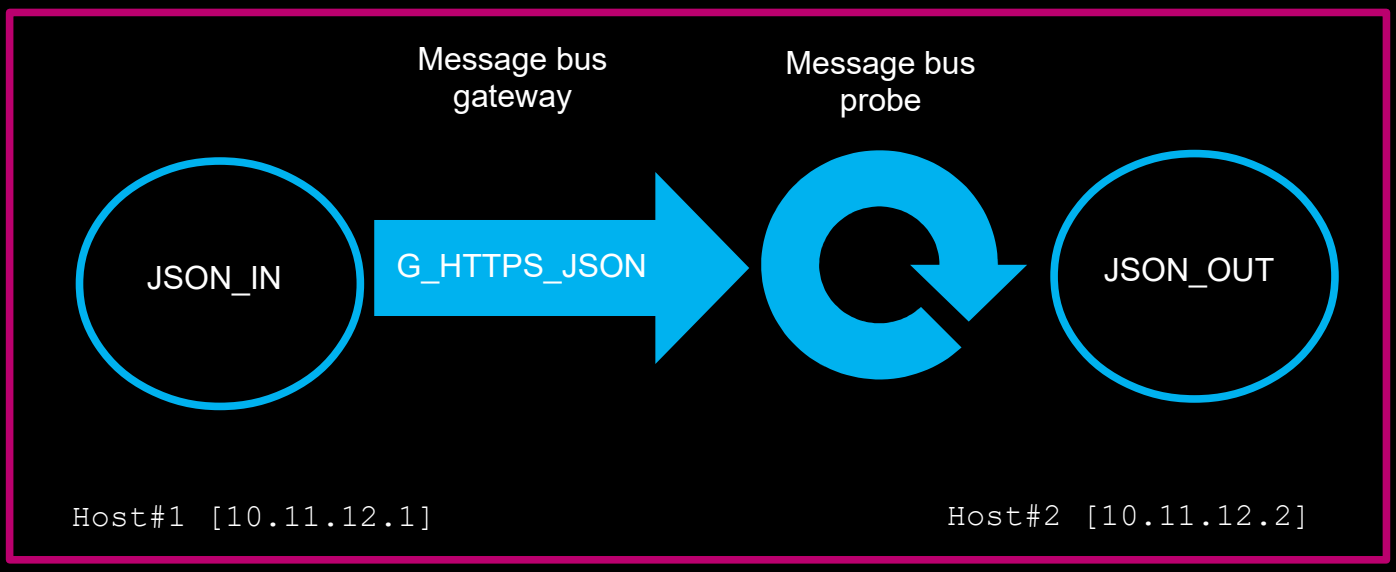
```
nco_g_xml -propsfile ./G_HTTPS.props > /tmp/G_HTTPS.ssl.log
```

Reading the log:

Successful SSL handshakes

```
grep -i found G_HTTPS.ssl.log
Found trusted certificate:


grep '\*\*\*' G_HTTPS.ssl.log
*** ClientHello, TLSv1.2
*** ServerHello, TLSv1.2
*** Certificate chain
*** ECDH ServerKeyExchange
*** ServerHelloDone
*** ECDHClientKeyExchange
*** Finished
*** Finished
*** ClientHello, TLSv1.2
*** ServerHello, TLSv1.2
*** Finished
*** Finished
*** ClientHello, TLSv1.2
*** ServerHello, TLSv1.2
*** Finished
*** Finished
```

14 Example Secure HTTP Transport with JSON



The Message bus Gateway can send secure JSON messages to the Message bus probe using HTTPS. In this example configuration the Message bus probes SSLWEBHOOK tools were used to create the certificates. The example configuration illustrates how to test the XML gateways requirements, and how to debug SSL communications.

The example configuration is for Red Hat Linux where platform is set to linux2x86.

In this example, the scripts from the HTTPS listener FAQ are used to create the certificates used, as they provide Subject Alternate Name configuration features seen in production certificates.

Using the message bus probe configuration as a HTTPS listener
https://www.ibm.com/support/pages/using-message-bus-probe-configuration-https-listener

14.1 Creating the SSL certificates

The HTTPS listener FAQ for the message bus probe includes a set of scripts for the creation and testing of the message bus port.

```
createCAWebhookSSL.sh  - Creates a CA certificate and a CA signed webhook SAN certificate
createWebhookSSL.sh    - Creates a self-signed webhook SAN certificate
check_CACERTS.sh       - Lists the CACERT's directories certificates and keystores
check_ssl_json_port.sh - Tests the message bus probes HTTPS listener port
```

14.1.1 Creating a certificate for the Message bus probe

Usage:

```
./createCAWebhookSSL.sh
Usage : createCAWebhookSSL.sh [Webhook FQDN] [Webhook EXT]
Where [Webhook EXT] is like
 SAN=dns:test.company.com,ip:192.168.20.20
```

In the example, the message bus probe server is host#2.
To create a suitable CA certificate and CA signed certificate with all the useful SAN details, in the domain *.ibm.com.

```
./createCAWebhookSSL.sh host#2.ibm.com
SAN=dns:localhost,ip:127.0.0.1,dns:host#2.ibm.com,dns:host#2,ip:10.11.12.2
```

The default password is 'netcool' for the scripts and this needs to be used when prompted.

The main files created in CACERTS are.

```
KeyStore.jks           - Used by the message bus probe
TrustStore.jks
cakey.pem
careq.pem
caroot.cert            - CA certificate
caroot.srl
certificate.conf
rand
webhook.cert
webhook.pem            - Webhook certificate signed by the CA certificate
webhook.signed.cert
webhook.unsigned.cert
```

14.1.2 Creating a self-signed certificate for the XML Gateway

Usage:

```
./createWebhookSSL.sh

Usage : createWebhookSSL.sh [Webhook FQDN] [Webhook EXT]

Where [Webhook EXT] is like
 SAN=dns:test.company.com,ip:192.168.20.20
```

In the example, the XML gateway server is host#1.
To create a suitable self-signed certificate with all the useful SAN details, in the domain *.ibm.com.

```
./createWebhookSSL.sh host#1.ibm.com
SAN=dns:localhost,ip:127.0.0.1,dns:host#1.ibm.com,dns:host#1,ip:10.11.12.1
```

The default password is 'netcool' for the scripts and this needs to be used when prompted.

The main files created in SSCERTS are.

```
KeyStore.jks    - Used by the XML Gateway as GatewayKeystore.jks
webhook.cert
webhook.pem     - Webhook self-signed certificate imported in the probes KeyStore.jks
```

14.1.3 Importing the message bus probes certificates

The XML Gateway can just use the message bus probes KeyStore.jks.

In order to illustrate the certificate requirements, the self-signed certificates keystore is used, which requires the message bus probes CA and webhook certificate imported, to create trust.

For ease, the file system is assumed to be shared between host#1 and host#2.

```
cd $NCHOME/omnibus/gates/G_HTTPS_JSON/JKS
cp $NCHOME/omnibus/utils/SSLWEBHOOK/SSCERTS/KeyStore.jks  GatewayStore.jks
cp $NCHOME/omnibus/utils/SSLWEBHOOK/CACERTS/caroot.cert .
cp $NCHOME/omnibus/utils/SSLWEBHOOK/CACERTS/webhook.pem .

keytool -import -keystore GatewayStore.jks -alias targetrootca -file ./caroot.cert
-storepass netcool
keytool -import -keystore GatewayStore.jks -alias targetwebhook -file ./webhook.pem
-storepass netcool
```

Checking the keystores contents

```
keytool -list -keystore GatewayStore.jks -storepass netcool
webhook, PrivateKeyEntry,
targetrootca, trustedCertEntry,
targetwebhook, trustedCertEntry,
```

To check the details us the '-v' option.

```
keytool -v -list -keystore GatewayStore.jks -storepass netcool
```

14.1.4 Importing the XML gateways certificate

```
cd $NCHOME/omnibus/utils/SSLWEBHOOK/CACERTS
keytool -import -keystore KeyStore.jks -alias webhookssc -file ../SSCERTS/webhook.pem
-storepass netcool
```

Checking the keystores contents

```
keytool -list -keystore KeyStore.jks -storepass netcool
webhookssc, trustedCertEntry,
webhook, PrivateKeyEntry,
```

To check the details us the '-v' option.

```
keytool -v -list -keystore KeyStore.jks -storepass netcool
```

14.2 Message bus probe configuration

14.2.1 Property file

```
File : message_bus.props-webhook-GW-8443-json

Server                  : 'JSON_OUT

# Transport

Port                    : 8443

TransportType           : 'Webhook'
TransportFile : '$NCHOME/omnibus/probes/linux2x86/GW-webhook-8443.properties'
# Transformer
MessagePayload          : 'json'
JsonMessageDepth        : 10
TransformerFile         : ''

# SSL/TLS configuration
EnableSSL               : 'true'
KeyStore                : '$NCHOME/omnibus/utils/SSLWEBHOOK/CACERTS/KeyStore.jks'
KeyStorePassword        : 'netcool'
#
RulesFile               : '$NCHOME/omnibus/probes/linux2x86/discard.rules'
#
# Best practice setting to prevent and detect problems
NetworkTimeout          : 15
PollServer              : 60
# Buffering
Buffering               : 1
BufferSize              : 200
FlushBufferInterval     : 9
# Performance tuning
DisableDetails          : 1
# Heartbeating
HeartbeatInterval       : 0
ProbeWatchHeartbeatInterval: 60
# Queue sizes
MaxEventQueueSize       : 50000
TransportQueueSize      : 50000
#EOF
```

14.2.2 Transport properties

```
File: GW-webhook-8443.properties

webhookURI=/
validateRequestURI=OFF
idleTimeout=20
# EOF
```

14.2.3 Discard rules

```
File: discard.rules
discard
# EOF
```

14.3 Testing the message bus probe

Run the message bus probe from the coommand line, and send a test message to the probes port using curl or wget.

```
$NCHOME/omnibus/probes/nco_p_message_bus -propsfile
$NCHOME/omnibus/probes/linux2x86/message_bus.props-webhook-GW-8443-json -messagelevel
debug -messagelog stdout
```

To send a message using the script provided with the HTTPS listener FAQ.

```
cd $NCHOME/omnibus/utils/SSLWEBHOOK
./check_ssl_json_port.sh localhost 8443 CACERTS/caroot.cert
```

To test using the FQDN, as the XML gateway uses.

```
./check_ssl_json_port.sh host#2.ibm.com 8443 CACERTS/caroot.cert
```

To test from the XML gateway server, using curl.
Insecure.

```
curl -k -v -H "Content-Type: application/json" -d '{"test":"Message text"}' -X POST
https://host#2.ibm.com:8443/
```

Using the CA certificate.

```
cd $NCHOME/omnibus/gates/G_HTTPS_JSON/JKS
curl --cacert ./caroot.cert -v -H "Content-Type: application/json" -d '{"test":"Message
text"}' -X POST https://host#2.ibm.com:8443/
```

One the Message bus probe logs the received test message, start on configuring the XML gateway.

14.4 XML Gateway configuration

14.4.1 Property file

```
File:  G_HTTPS_JSON.props

Gate.Reader.Server                    : 'JSON_IN'
Name                                  : 'G_HTTPS_JSON'
###
# Log level
MessageLevel                          : 'debug'
#MessageLevel                          : 'info'
###
# files
MessageLog                            : '$OMNIHOME/log/G_HTTPS_JSON.log'
Gate.MapFile                          : '$OMNIHOME/gates/G_HTTPS_JSON/xml.map'
Gate.StartupCmdFile                   :
'$OMNIHOME/gates/G_HTTPS_JSON/xml.startup.cmd'
Gate.Reader.TblReplicateDefFile       :
'$OMNIHOME/gates/G_HTTPS_JSON/xml.reader.tblrep.def'
###
# Transport
Gate.XMLGateway.TransportType         : 'HTTP'
Gate.XMLGateway.TransportFile         :
'$OMNIHOME/gates/G_HTTPS_JSON/httpsTransport.properties'
###
# Transformer
Gate.XMLGateway.TransformerInputType  : 'JSON'
Gate.XMLGateway.MessageID             : 'netcoolEvents'
Gate.XMLGateway.TransformerFile       :
'$OMNIHOME/gates/G_HTTPS_JSON/httpstransformers.xml'
Gate.Reader.IducFlushRate             : 11
###
# Debugging
Gate.NGtkDebug                        : TRUE
Gate.Java.Debug                       : TRUE
Gate.Mapper.Debug                     : TRUE
Gate.Reader.Debug                     : TRUE
Gate.XMLGateway.Debug                 : TRUE
Gate.XMLGateway.PublishTraceOn        : TRUE
####
# Uncomment for SSL debug logging
#Gate.Java.Arguments                   : '-Djavax.net.debug=ssl:handshake:verbose'
# EOF
```

14.4.2 Transport properties

```
File: httpsTransport.properties

###
# Target listener
clientURL=https://<FQDN>:8443/
###
# SSL certificate stores
trustStore=/opt/IBM/tivoli/netcool/omnibus/gates/G_HTTPS_JSON/JKS/GatewayStore.jks
keyStore=/opt/IBM/tivoli/netcool/omnibus/gates/G_HTTPS_JSON/JKS/GatewayStore.jks
trustStorePassword=netcool
keyStorePassword=netcool
#
bufferSize=1
# EOF
```

14.4.3 Transformers file

```
File : httpstransformers.xml

<?xml version="1.0" encoding="UTF-8"?>
<tns:transformers
        xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool/transformer"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<tns:transformer name="netcoolEvents" type="northbound"
endpoint="https://<FQDN>:8443/"
className="com.ibm.tivoli.netcool.integrations.transformer.EmptyTransformer">
        </tns:transformer>
</tns:transformers>
```

14.4.4 Table replication

```
File: xml.reader.tblrep.def

###
# Replicate with a filter and after-iduc
REPLICATE ALL FROM TABLE 'alerts.status'
USING MAP 'StatusMap'
FILTER WITH 'Poll=1'
AFTER IDUC DO 'Poll=2'
;
#EOF
```

14.4.5 Mapping file

```
File: xml.map

CREATE MAPPING StatusMap
(
        'Identifier'        =    '@Identifier'           ON INSERT ONLY,
        'Node'              =    '@Node'                 ON INSERT ONLY,
        'NodeAlias'         =    '@NodeAlias'            ON INSERT ONLY
                                                                NOTNULL '@Node',
        'Manager'           =    '@Manager'              ON INSERT ONLY,
        'AlertGroup'        =    '@AlertGroup'           ON INSERT ONLY,
        'AlertKey'          =    '@AlertKey'             ON INSERT ONLY,
        'Severity'          =    '@Severity',
        'Summary'           =    '@Summary',
        'FirstOccurrence'   =    '@FirstOccurrence'      ON INSERT ONLY,
        'LastOccurrence'    =    '@LastOccurrence',
        'Type'              =    '@Type'                 ON INSERT ONLY,
        'Tally'             =    '@Tally',
        'ServerName'        =    '@ServerName'           ON INSERT ONLY,
        'ServerSerial'      =    '@ServerSerial'         ON INSERT ONLY
);
# EOF
```

14.4.6 Start-up file

```
xml.startup.cmd
<empty>
```

14.5 Testing the XML gateway

The XML gateway needs to find a valid event to send the JSON message to the message bus probe.

14.5.1 Start the message bus probe

Start the message bus probe from the command line, before running the XML gateway, if it's not already running.

```
$NCHOME/omnibus/probes/nco_p_message_bus -propsfile
$NCHOME/omnibus/probes/linux2x86/message_bus.props-webhook-GW-8443-json -messagelevel
debug -messagelog stdout
```

14.5.2 Start the XML gateway

Start the XML gateway from the command line.

```
Cd $NCHOME/omnibus/gates/G_HTTPS_JSON
nco_g_xml -propsfile ./G_HTTPS_JSON.props -messagelevel debug -messagelog stdout
```

14.5.3 Insert test event script

File: insert_test_event.sh

```
#! /bin/sh
if [ $# -ne 1 ]
then
echo "Usage : `basename $0` NCOMS"
exit
fi
# Requires NCHOME to be set and the user/password
SQLCMD="$NCHOME/omnibus/bin/nco_sql -server ${1} -user root -password "
export SQLCMD DATE
DATE=`date`

$SQLCMD '' <<EOF
insert into alerts.status
(Identifier,Severity,Summary,AlertGroup,AlertKey,OwnerUID,OwnerGID,Poll)
values  ('test-alert-${DATE}',4,'TEST event from
G_HTTPS_JSON','testing','G_HTTPS_JSON',65534,0,1);
go
quit
EOF

#EOF
```

14.5.4 Insert a test event

Insert a test event into the JSON_IN object server using the example script.

```
cd $NCHOME/omnibus/gates/G_HTTPS_JSON
./insert_test_event.sh JSON_IN
```

The event can be entered manually into the object server JSON_IN using nco_sql.

```
e.g.
nco_sql -server JSON_IN -user root
Password:
(Identifier,Severity,Summary,AlertGroup,AlertKey,OwnerUID,OwnerGID,Poll)
values  ('test-alert-1',4,'TEST event from
G_HTTPS_JSON','testing','G_HTTPS_JSON',65534,0,1);
go
```

The Identifier needs to be unique for each insert ['test-alert-1'].

14.5.5 Successful event transmission

The XML gateways logging.

```
Debug: D-GJA-000-000: [ngjava]: XMLGateway: Transforming message - ...
Debug: D-GJA-000-000: [ngjava]: XMLGateway: Transformed message - ...
Debug: D-GJA-000-000: [ngjava]: XMLGateway: Batch Manager adding message to target ...
Debug: D-GJA-000-000: [ngjava]: XMLGateway: Creating URL connection to
'https://host#2.ibm.com:8443/'.
Debug: D-GJA-000-000: [ngjava]: XMLGateway: Server response code = 204. [No Content]. Payload:
```

The message bus probes logging.

On start-up the probe logs.

```
Information: I-JPR-000-000: Webhook URL:https://host#2:8443/
```

On reading the test message the probe logs.

```
[Event Processor] test:      Message text
```

On reading an event from the gateway the probe logs and event.

```
[Event Processor] resync_event:      false
[Event Processor] Identifier:        test-alert-<timestamp>
[Event Processor] ServerName:        JSON_IN
[Event Processor] ServerSerial:      414
[Event Processor] Summary:   TEST event from G_HTTPS_JSON
[Event Processor] AlertGroup:        testing
[Event Processor] AlertKey:  G_HTTPS_JSON
[Event Processor] Tally:     1
[Event Processor] NodeAlias:
[Event Processor] Node:
[Event Processor] Severity:  4
[Event Processor] FirstOccurrence:   <timestamp>
[Event Processor] LastOccurrence:    <timestamp>
[Event Processor] Manager:
[Event Processor] Type:      0
```

14.6 Troubleshooting

XML Gateway logs an error when attempting to send HTTPS message.

```
Caught IOException when writing to output stream:
java.security.NoSuchAlgorithmException: Error constructing implementation
```

Cause : Missing JKS file[s] definition.

SSL debug message from gateway log:

```
SSLSocket duplex close failed (
"throwable" : {
  java.net.SocketException: Socket is closed
```

Cause : Server already closed the socket, and is expected for the message bus probe. Check to confirm the event was read successfully.

New HTTP,transport properties file setting.

```
useNullHostnameVerifier

Use this property to disable hostname verification in the HTTP client against the web
server's SSL certificate.
This unblocks the authentication exception caused by the hostname in a valid SSL
certificate being different from the web server's current hostname.
The default is false.
```