# Tivoli Netcool Support's Guide to the Message bus probe Websocket and Webhook

by

Jim Hutchinson

Document release: 3.3



# Table of Contents

**1Introduction ..... 4**

1.1Overview ..... 4
1.2General Transport features ..... 5
1.2.1Websocket ..... 5
1.2.2Webhook ..... 6
1.2.3WebhookV2 ..... 7
1.3Request features ..... 8
1.3.1Login ..... 8
1.3.2Resync ..... 9
1.3.3Subscribe ..... 9
1.3.4autoReconnect ..... 10
1.3.5WebSocket Features ..... 10
1.3.6OAuth feature ..... 10
1.3.7Plus-Plus tokens ..... 11
1.4New Integrations ..... 12
1.4.1Request for Enhancement ..... 12
1.5Initial Checks ..... 12

**2Message Bus Probe Configuration ..... 13**

2.1Message bus probe example configurations ..... 14
2.1.1iDirect_pulse [WebSocket] ..... 14
2.1.2Prometheus [Webhook] ..... 15
2.1.3Logstash [Webhook] ..... 15

**3WebSocket Transport ..... 16**

3.1.1Simple WebSocket subscription ..... 19
3.2Ciena MCP Example ..... 20
3.2.1get_ssl.sh ..... 21

**4Webhook Transport ..... 22**

4.1Simple Webhook Example ..... 25
4.1.1Property file ..... 25
4.1.2Transport properties file ..... 25
4.1.3Send event script ..... 25
4.1.4Sending an event ..... 26
4.2Basic Authentication Example ..... 27
4.2.1Property file: ..... 27
4.2.2Transport Properties file ..... 27
4.2.3Sending events ..... 28
4.3SSL Example ..... 29
4.3.1Property file: ..... 29
4.3.2Transport Properties file ..... 29
4.3.3Sending Events ..... 30
4.4Secure Webhook – HTTPS and Basic Authentication ..... 32
4.4.1Property file ..... 32
4.4.2Transport properties ..... 33
4.4.3Property and properties password encryption ..... 34
4.5Custom HMAC Signature security ..... 35
4.5.1Probe property file ..... 35
4.5.2Transport properties file ..... 35
4.5.3Probe rules file ..... 35
4.5.4Probe parser file ..... 36
4.5.5Pre-parser Java Script ..... 37
4.5.6HMAC signature test script ..... 38
4.6Webhook OAuth example from the manual ..... 39
4.6.1Probe Property file ..... 39

IBM Copyright 2026 

4.6.2Transport Properties....39
4.7Cisco EPNM Custom example....40
4.7.1Probe property....40
4.7.2Transport properties:....41
4.8Cisco APIC probe....42
4.8.1message_bus_cisco_apic.props....43
4.8.2CiscoAPIC.transport.properties....44
4.9Resynchronisation example....45
4.9.1Probe property file....45
4.9.2Transport properties file....45
4.9.3Rules file logic....46
4.10Webhook with multiple parsing example....47
4.10.1Property file....47
4.10.2Transport properties file....47
4.10.3Parser file....48
4.10.4Send an event to a URI script....49
4.10.5Example Results....49
4.10.6Sending an event....50
4.10.7endpoint.rules....50
4.11Webhook and exploding event array example....51
4.11.1Property File....51
4.11.2Transport properties....51
4.11.3Testing....51
4.11.4send_file_to_uri.sh script....52
4.11.5Explode tokens to arrays rules....53
4.12WebSocket connecting to STOMP....55
4.12.1Property file....55
4.12.2Transport properties....56
4.12.3Transformer file....56
4.12.4Java Script Pre-Parser....57

**5WebhookV2 transport .... 58**

5.1GLOBAL....59
5.2LOGIN....59
5.2.1GET_ACCESS_TOKEN....59
5.2.2GET_REFRESH_ACCESS_TOKEN....59
5.3 OAUTH....60
5.4OIDC....60
5.4.1INTROSPECTION....60
5.5 SUBSCRIBE....61
5.5.1GET_SUBSCRIPTION....61
5.5.2GET_SUBSCRIPTION_REFRESH....61
5.6 RESYNC....61
5.6.1RESYNC_CHILD_BY_FDN....61
5.7 LOGOUT....62
5.7.1REVOKE_ACCESS_TOKEN....62

**6The Probe Environment file .... 63**

6.1Custom Java....63
6.2Proxy settings....63
6.3Enforcing Ipv4 or IPv6....64

**7Evaluating a REST service .... 65**

7.1Host and Port....65
7.2SSL details....66
7.3Curl check....67

**8Troubleshooting .... 68**

8.1Checking CURL commands....68
8.2Debugging SSL Connectivity....69
8.3Creating ProbeKeyStore.jks....70

8.3.1Setting the probes truststore....71
8.4Transport debug log....72
8.5Unique Transport debug log....73
8.6Reviewing the probes log....74
8.7Parsing events....75
8.7.1JSON Nested events....75
8.8Useful FAQs....76
**9Known Issues....77**
9.1DT465640 - Message bus probe - Webhook transport - Webhook accepts methods other than POST....77
9.2DT400518 - Message bus probe Webhook event size cannot be controlled via the transport properties....77

# 1 Introduction

## 1.1 Overview

The Message bus probe is a diverse probe which can be used for many Message bus integrations. This document will discuss exclusively the integration of the probe using its Websocket and Webhook facilities. Please refer to the Support's guide to the Message Bus integration for an overview of the probes design and for descriptions of the probes other functionality.

The Message bus probe accepts XML and JSON event formats. In order to keep the examples simple, only the JSON event payload will be considered within this document. Refer to the message bus probe manual and the Support's guide to the Message Bus integration for general XML event processing.

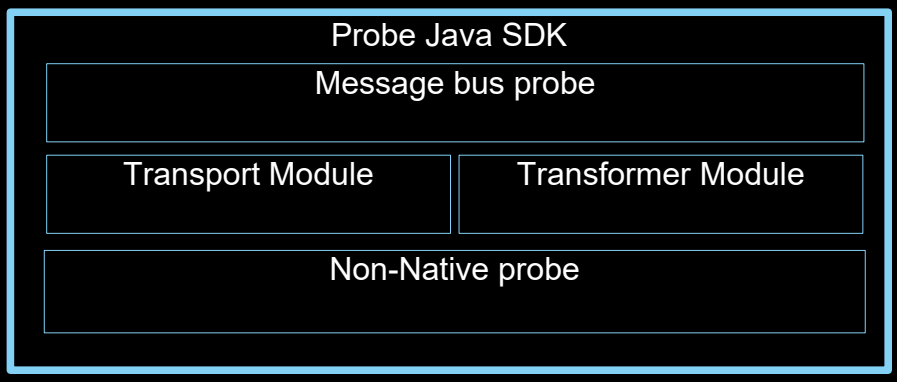
The Websocket transport definition does not include a Webhook listener, and the Webhook transport does not include the Websocket transport connections, but both transports include common login, logout, resynchronisation and subscription features. The example transport properties are provided in the directory $NCHOME/omnibus/java/conf/default. It is recommended to never modify the files in the $NCHOME/omnibus/java/conf as updates from probes packages will not take affect. Refer to the default directory for template files after installing any of the supporting probe packages to check to see if new features are available.

The Websocket Transport is used in a number of specific probes including the CISCO APIC probe. Check the probe manual and configuration files of other specific probe Websocket integrations when attempting to integrate the Message Bus probe with a new event managers REST API. Be aware that such integrations exist because the Message Bus probe is not compatible with these systems, due to vendor specific requirements.

Most event manager vendors that use REST API's adhere to open standards, and will therefore be accessible using the Message bus probe, provided the Message bus probe includes the required functionality. When the Message Bus probe does not include the required functionality, it is recommended that a Request for Enhancement is raised. It is recommended to raise an RFE whenever a new integration is required, so that the Product Managers are aware of the number of customers using specific event managers.

Where possible integration steps for popular event management systems are included in the Message Bus probe product documentation.

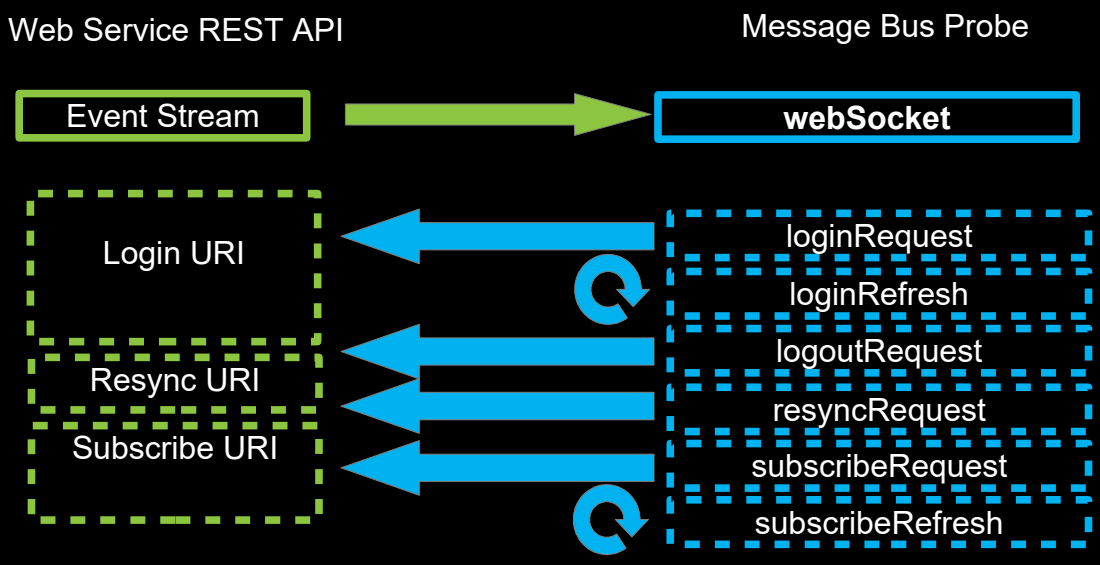
The Message bus probe is made up of four probe packages grouped together using the Probe Java SDK package:



## 1.2 General Transport features

### 1.2.1 Websocket

The Websocket Transport provides a method to connect to a WebSocket stream [ws://] and receive events. The WebSocket is limited due to the use of Host/Port property settings which fixes the target host and port:

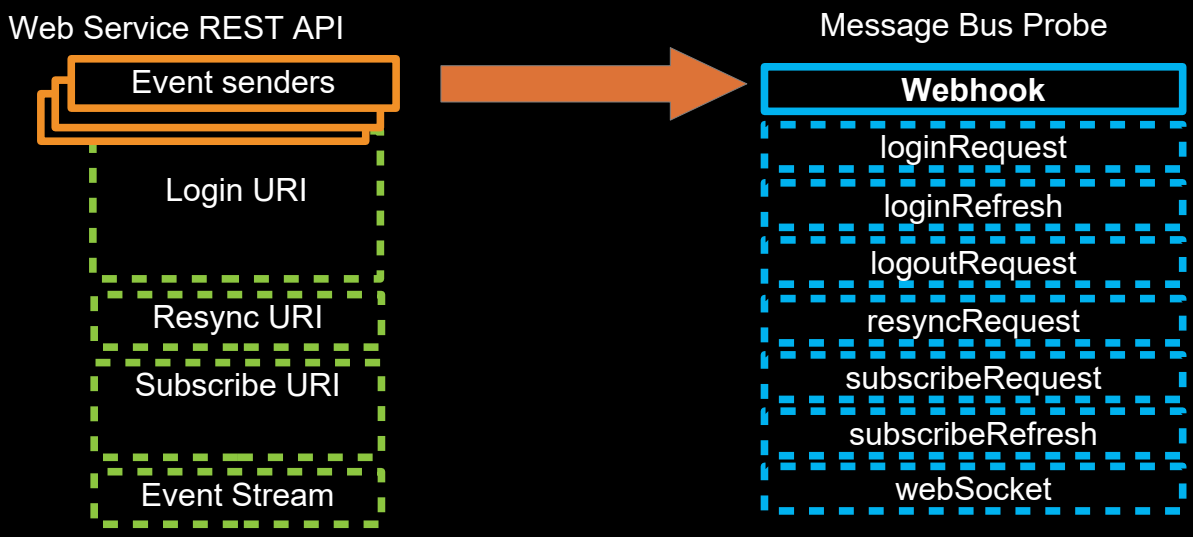



The WebSocket transport now has an extra set of transport properties, that allow for an action after sending the LoginRequest. Which useful when a token is required for the post request, and this token is obtained upon login.

```
postLoginRequestURI=
postLoginRequestMethod=
postLoginRequestContent=
postLoginRequestHeaders=
```

### 1.2.2 Webhook

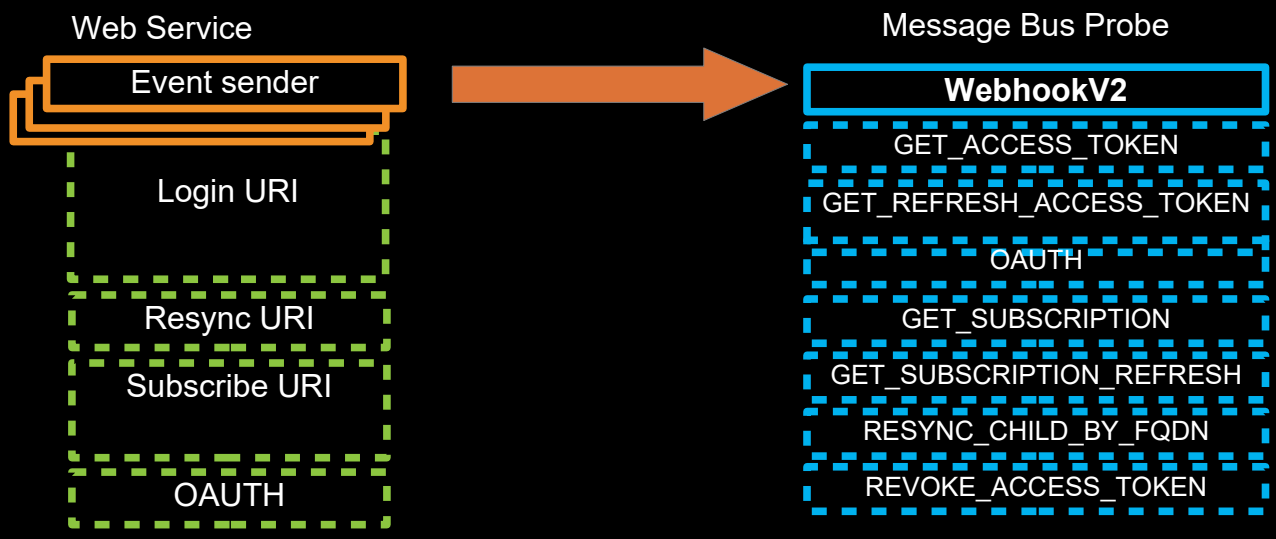
The Webhook Transport provides a Webhook listener for sending events to, as well as the login,resynch and subscrbe facilities if required.



### 1.2.3 WebhookV2

The WebhookV2 Transport provides a multichannel solution for the Webhook transport. Rather than using the transport properties file to define a loosely bound set of activities, the file refers to a multichannel transport properties file.

```
"GLOBAL":
"LOGIN":
     "GET_ACCESS_TOKEN":
     "GET_REFRESH_ACCESS_TOKEN":
"OAUTH":
"SUBSCRIBE":
     "GET_SUBSCRIPTION":
     "GET_SUBSCRIPTION_REFRESH":
"RESYNC":
     "RESYNC_CHILD_BY_FDN":
"LOGOUT":
     "REVOKE_ACCESS_TOKEN":
```



## 1.3 Request features

The request features have three main types.

- login|logout|refresh
- resynchronisation
- subscription|refresh

The message bus probe does not wait for one request to complete, before moving onto the next request, apart from the postLoginRequest.

### 1.3.1 Login

These settings are used to construct the HTTP request to login to the server.
Typically a token is obtained using the keepTokens property, which is used with other requests.
If the token obtained during login, is essential, and required for a next request, the postLoginRequest needs to be used.

```
loginRequestURI=
loginRequestMethod=
loginRequestContent
loginRequestHeaders=
```

The postLoginRequest is sent after the loginRequest completes successfully.

```
postLoginRequestURI=
postLoginRequestMethod=
postLoginRequestContent=
postLoginRequestHeaders=
```

These settings are used to construct the HTTP request to periodically refresh the login session.
The requested period is specified by the property loginRefreshInterval.

```
loginRefreshURI=
loginRefreshMethod=
loginRefreshContent=
loginRefreshHeaders=
loginRefreshInterval=
```

These settings are used to construct the HTTP request sent before the probe disconnects from the server.
The request is used to unsubscribe or logout from the server.

```
logoutRequestURI=
logoutRequestMethod=
logoutRequestContent=
logoutRequestHeaders=
```

### 1.3.2 Resync

These settings are used to construct the HTTP request to get a set of events, periodically from the server
The requests period is defined by the ResyncInterval probe property.
Typically the resynchronisation request is for historical event data, and only collected on probe start-up.
In this case the probe property InitialResync is set to 'true', and ResyncInterval set to 0.
The InitialResync can be set to 'false', and ResyncInterval set to a value, such as 300, when some other request is required to complete before the synchronisation events are collected.

```
resyncRequestURI=
resyncRequestMethod=
resyncRequestContent=
resyncRequestHeaders=
```

### 1.3.3 Subscribe

The subscribe request is used to find the values of tokens listed in the keepTokens property.
The subscribeRequestURI can be used to inform the server where to send the events, or provide a subscription location for the WebSocket transport.
The Cisco APIC probe integration is an example of where the subscribe request is used to create a subscription and websocket event stream. With the websocket's URI defined by the value of a cookie returned by the server following a successful subscription.

Only with the resynchronisation request is event data collected from the server for event processing.

These settings are used to construct the HTTP request which collects the subscription details.
Usually after successfully connecting to the target device for WebSocket, or enabling the Webhook listener.
Sometimes keepTokens is used to collect a subscription identifier used to construct the connection path.

```
subscribeRequestURI=
subscribeRequestMethod=
subscribeRequestContent=
subscribeRequestHeaders=
```

These settings are used to construct the HTTP request to keep the subscription active.
The request is sent periodically as specified by the subscribeRefreshInterval setting.

```
subscribeRefreshURI=
subscribeRefreshMethod=
subscribeRefreshContent=
subscribeRefreshHeaders=

subscribeRefreshInterval=
```

### 1.3.4 autoReconnect

The autoReconnect feature allows the probe to reconnect to a service that it was disconnected from.
Set to "ON" to re-establish the connection to the HTTP server before sending a HTTP request.
Set to "OFF" to disconnect the probe after a connection close request is received from the HTTP server.
The default value is "OFF"

autoReconnect=ON

With the latest transport module, the keepTokens are preserved if the probe is disconnected from the HTTP server.

### 1.3.5 WebSocket Features

The WebSocket settings can be used to access one or two WebSockets, whose server is defined by the probes Host and Port property settings.

```
webSocketURI=<path#1>
webSocketPersistentURI=<path#2>
webSocketSubProtocol=

webSocketSubscribeMessage=

webSocketRefreshMessage=Ping
webSocketRefreshInterval=60
```

### 1.3.6 OAuth feature

OAuth features are available for authentication.

```
# OAuth2 Properties
tokenEndpointURI=<http://oauthserverhost:80/oauth/token>
# EITHER
# Basic Authentication
basicAuthenticationUsername=
basicAuthenticationPassword=
# OR
# Client secret
# clientId=
# clientSecret=
# scopes=
```

The OAuth authentication provides the token ++Oauth.access_token++ value.

This can then be used for the other settings using ++variable++ syntax.

```
httpHeader=Authorization=Bearer ++Oauth.access_token++,Accept=application/json,Content-
Type=application/json
subscribeRequestURI=/api/v1/faults
subscribeRequestMethod=POST
```

### 1.3.7 Plus-Plus tokens

The Plus-Plus tokens allow variables to be passed from the probe property file and event stream to the transport properties settings.

++Username++ taken from the probe property file Username
++Password++ taken from the probe property file Password
++user++ taken from the probe property file Username
++Host++ taken from the probe property file Host
++Port++ taken from the probe property file Port

++ProbeDisconnectTime++ taken from the timestamp in DataBackupFile
++OAuth.access_token++ taken from the OAuth token when connecting to tokenEndpointURI
++WebhookUrl++ taken from the full URL based on the webhookURI setting

++token++ taken from the returned value due to **"keepTokens=token"** setting

++{start, stop, step}++ is used for pagenation

**Custom ++variables++:**

Probe property file:

```
Cookie : "custom_token,another_token"
```

These can then be referred to in the transport properties file using

```
++custom_token++
++another_token++
```

## 1.4 New Integrations

For new integrations the vendor needs to provide adequate documentation to allow for successful Message bus probe configuration. The current limitations of the Message bus probe are documented in the product manual. These manuals include a  Document control page, which lists any new features. This can be used to check for any new properties.

### 1.4.1 Request for Enhancement

It is recommended that a Request for Enhancement is raised for any integration that is not presently documented. This allows Product Management to know which integrations are being used as well as understanding if there is a need to create a specific probe integration. Should the Message bus probe not be able to provide a suitable or incomplete solution, please update the RFE and list the Message bus probes deficiencies adding any relevant information that might assist in any new probe development. For all RFE's, the IBM Account Team are the correct contact for any follow-up, including arranging RFE reviews and estimated timescales.

## 1.5 Initial Checks

The vendor of the service should provide suitable documentation regarding how to connect to the service, which will include examples of the parameters passed to the service and the responses. Use the examples to confirm that the service being connected to behaves as documented and make notes of the specific variables being returned and steps performed to obtain the event data.

The typical steps are

- Login / Authentication [tokens]
- Resynchronisation
- Subscription
- Refresh Login / Credentials
- Logout / Expiration

Basic determination of the required steps for  a successful implementation

- Determine the required parameters for login and authentication, and check the returned data.
- Determine how the login is kept active.
- Determine how to successfully logout of the service.

Event management checks

- Resynchronisation filter – does it include a relative time based option
- Subscription filter syntax
- Event limitations
- Data size limitations
-

Identify the following specific details

- The protocols used like http,https,ws, etc.
- HTTP Header requirements
- TLS requirements
- HTTP Methods PUT, GET, POST, etc.
- The event syntax

# 2 Message Bus Probe Configuration

The Message bus probe uses a number of files to create events.

- Probe Property file
- Transport properties file
- JSON Parser file (optional)

It is best to create copies of the original files and modify these copied files, since the message bus probe shares files with the XML Gateway and the default files can be updated during a package installation. It is recommended that the tranport properties and any other configuration files are checked for compatibility after any related package installation.

**Probe Property file** :
`$NCHOME/omnibus/probes/<platform>/message_bus_integration.props`

**Probe transport properties file** :
`$NCHOME/omnibus/probes/<platform>/integrationWebhookTransport.properties`

Where the transport file is a copy of the transport file from $NCHOME/omnibus/java/conf/default:
`restWebSocketTransport.properties`
or
`restWebhookTransport.properties`

**Transformer File** [JSON]:
`${NCHOME}/omnibus/probes/<platform>/message_bus_integration_parser.json`

Where the transformer file is a copy of a suitable transformer file, for example $NCHOME/omnibus/<platform>/default/message_bus_parser_config.json.

**SSL configuration :**

For SSL there is the addition of the KeyStore file:

KeyStore
`$NCHOME/omnibus/probes/<platform>/integration_ssl/KeyStore.jks`

It is easier to create a separate directory in the SSL case, and have all the related SSL files created there.
e.g.
`$NCHOME/omnibus/probes/<platform>/mbp/integration`

## 2.1 Message bus probe example configurations

The Message bus probe includes a number of standard configurations which are covered in the main probe manual. These can be reviewed to understand how to integrate other systems using the message bus probe.

### 2.1.1 iDirect_pulse [WebSocket]

This configuration connects to remote websockets given by the settings Host, Port, webSocketURI and webSocketPersistentURI. The configuration includes a subscription request and authentication via HTTP headers

File : message_bus_iDirect_pulse.props

```
Manager                 : 'iDirect_Pulse'
MessageLog              : '$OMNIHOME/log/message_bus_iDirect_pulse.log'
RulesFile               : '$OMNIHOME/probes/linix2x86/message_bus_iDirect_pulse.rules'
# Web Socket configuration
TransportType           : 'WebSocket'
Host                    : '<Server>'
Port                    : 80
TransportFile           : '${OMNIHOME}/probes/linix2x86/iDirectPulseTransport.properties'
Cookie                  : 'sid'
MessagePayload          : 'json.payload'
JsonNestedPayload       : 'json.data'
# Retry
RetryCount              : 3
RetryInterval           : 30
DataBackupFile          : '${OMNIHOME}/var/message_bus_pulse.backup'
# Authentication
Username                : '<username>'
Password                : '<password>'
#EOF
```

File : ${OMNIHOME}/probes/linux2x86/iDirectPulseTransport.properties

```
httpVersion=1.1
responseTimeout=15
# Authentication through HTTP Header
httpHeaders=Authorization=Basic ++Username++:++Password+
+,Accept=application/json,Content-Type=application/json,Use-Cookie=true,User-
Agent=IBM Netcool/OMNIBus Message Bus Probe
# Subscribing
subscribeRefreshURI=/api/1.0/core/cluster_info
subscribeRefreshMethod=GET
subscribeRefreshContent=
subscribeRefreshInterval=300
# WebSocket
webSocketURI=/api/1.0/dde/alarm?start_time__gte=0
webSocketPersistentURI=/api/1.0/dde/alarm?start_time__gte=++ProbeDisconnectTime++
# Keep alive
webSocketRefreshMessage=netcoolping
webSocketRefreshInterval=60
refreshRetryCount=5
# If SSL required
#securityProtocol=TLSv1.2
#EOF
```

Creates a WebSocket:
**ws:<Server>:80/api/1.0/dde/alarm?start_time__gte=0**

### 2.1.2 Prometheus [Webhook]

This configuration creates a Webhook [http] listener on the probe server as defined by the Host, Port and webhookURI settings.

File : message_bus_prometheus.props

```
RulesFile       : '$OMNIHOME/probes/linux2x86/message_bus_prometheus.rules'
TransformerFile: '${OMNIHOME}/probes/linux2x86/message_bus_prometheus_parser.json'
TransportFile   : '${OMNIHOME}/probes/linux2x86/prometheusWebhookTransport.properties'
TransportType   : 'Webhook'
Host            : '<FQDN>'
Port            : 8080
```

File : ${OMNIHOME}/probes/linux2x86/prometheusWebhookTransport.properties

```
webhookURI=/probe/webhook/prometheus
idleTimeout=180
#EOF
```

Creates a webhook listener:
`http://<FQDN>:8080/probe/webhook/prometheus`

### 2.1.3 Logstash [Webhook]

This configuration creates a Webhook [http] listener on the probe server as defined by the Host, Port and webhookURI settings.

File : message_bus_logstash.props

```
RulesFile         : '$OMNIHOME/probes/linux2x86/message_bus_logstash.rules'
TransformerFile   : '${OMNIHOME}/probes/linux2x86/message_bus_logstash_parser.json'
TransportFile     : '${OMNIHOME}/probes/linux2x86/logstashWebhookTransport.properties'
TransportType     : 'Webhook'
Host              : '<FQDN>'
Port              : 8080
```

File : ${OMNIHOME}/probes/linux2x86/logstashWebhookTransport.properties

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

Creates a webhook listener:
`http://<FQDN>:8080/probe/webhook/logstash`

# 3 WebSocket Transport

See highlighted properties for WebSocket specific transport properties.

The Message bus probe comes with an example WebSocket transport properties file:
Directory : $NCHOME/omnibus/java/conf/default
File : **restWebSocketTransport.properties**

```
# Specifies the HTTP version. Supports version 1.1 (default) or 1.0.
#httpVersion=1.1

# A comma-seperated list of HTTP headers to be set in all
# outgoing HTTP request.
#httpHeaders=

# The timeout (in seconds) to wait for the HTTP response
# from the server. Default is 60 seconds.
#responseTimeout=60

# A comma-separated list of attributes to extract from the header and
# JSON body received in a HTTP response.
#keepTokens=

# Specifies the SSL protocol. Example : TLSv1.2
#securityProtocol=

####
# OAuth2 Properties
# - The supported grant type is Resource Owner Password Crendentials Grant.
# The Username and Password property will be used as the
# resourceOwnerUsername and resourceOwnerPassword to obtain the
# access token. Please ensure these probe properties are set.
##

# The token endpoint to request for an access token.
# Default is unset (disabled).
# Specify the path on the remote host or the
# the full token endpoint URL to request for an
# access token. For example: tokenEndpointURI=/oauth/token
# or tokenEndpointURI=http://oauthserverhost:80/oauth/token
#tokenEndpointURI=

# Specifies the basic authentication credentials.
# As an alternative for servers that does not support basic authentication,
# use clientId and clientSecret properties to send the client
# credentials in the request-body.
#basicAuthenticationUsername=
#basicAuthenticationPassword=

# Specifies the client credentials to send in the request-body
# as the "client_id" and "client_secret" attributes.
#clientId=
#clientSecret=

# Specifies the access scopes. Accepts a comma-separated list.
# For example: read,write
```

```
####
# Each HTTP request can be configured by the RequestURI, RequestMethod,
# RequestContent and RequestHeader.
# 1. Specify the path in the RequestURI property. Example : /login
# 2. Specify the HTTP Method in the RequestMethod. Example : POST, GET, PATCH, etc.
# 3. (Optional) Specify the HTTP body in the RequestContent. Example:
{\"sample\":\"json\"}
# 4. (Optional) Specify the additional HTTP headers in the RequestHeaders property. This
#    will override any headers set in the httpHeader property.
# 5. Specify the interval (in seconds) for loginRefresh and subscribeRefresh request
property
#    to enable a HTTP request to be sent periodically.
####
# Constructs the HTTP request which will be sent after the OAuth
# access token request.
#loginRequestURI=
#loginRequestMethod=
#loginRequestContent=
#loginRequestHeaders=

# Constructs another HTTP request which will be sent after the above
# login request and before initiating websocket server.
#postLoginRequestURI=
#postLoginRequestMethod=
#postLoginRequestContent=
#postLoginRequestHeaders=

# Constructs the HTTP request which will be sent periodically
# as specified by the loginRefreshInterval.
#loginRefreshURI=
#loginRefreshMethod=
#loginRefreshContent=
#loginRefreshHeaders=
#loginRefreshInterval=

# Constructs the HTTP request which will be sent before disconnecting.
# It can be used to unsubscribe or logout from the system.
#logoutRequestURI=
#logoutRequestMethod=
#logoutRequestContent=
#logoutRequestHeaders=

# Constructs the HTTP request which will be sent periodically.
# The interval is configured by the ResyncInterval probe property.
#resyncRequestURI=
#resyncRequestMethod=
#resyncRequestContent=
#resyncRequestHeaders=

# Constructs the HTTP request which will be sent in the subscribe state
# after successfully connecting to the target device.
#subscribeRequestURI=
#subscribeRequestMethod=
#subscribeRequestContent=
#subscribeRequestHeaders=

# Constructs the HTTP request which will be sent periodically
# as specified by the subscribeRefreshInterval.
#subscribeRefreshURI=
#subscribeRefreshMethod=
#subscribeRefreshContent=
#subscribeRefreshHeaders=
#subscribeRefreshInterval=
```

```
# Specifies the path on the websocket server to connect.
#webSocketURI=

# Specifies the persistent path on the websocket server to connect.
# If this is set, it will be the first URI used but if the websocket handshake
# fails, then the webSocketURI is used.
#webSocketPersistentURI=

# Specifies the websocket subprotocol. (Optional)
#webSocketSubProtocol=

# Specifies the message to be included in the Websocket TextFrame and sent
# to the websocket server. (Optional)
#webSocketSubscribeMessage=

# Specifies the message to be included in the Websocket TextFrame and sent
# to the websocket server periodically. The websocketRefreshInterval (in seconds)
# specifies the interval to send this message. (Optional)
#webSocketRefreshMessage=
#webSocketRefreshInterval=

# Specifies the maximum payload frame length (in bytes) of the incoming websocket
message.
# Default is 65536 bytes.
#webSocketMaxFramePayloadLength=65536

# The number of retries for the refresh requests. Default is 0 (no retries).
#refreshRetryCount=0
```

### 3.1.1 Simple WebSocket subscription

The Host and Port are defined in the probe property file. These property settings along with the WebSocket URI's define the full URI.

e.g.

```
Host : 'localhost'
Port : 8080
webSocketURI=/websocket
```

Will result is a websocket call to the URI:

```
WebSocket URI: ws://localhost:8080/websocket
```

The probe can then receive events from this socket.

Subscribing to receive notifications using WebSocket

```
webSocketURI=/websocket
# Adding authentication
httpHeaders=Authorization=Basic ++Username++:++Password++,Content-Type=application/json
# Keeping the socket alive
webSocketRefreshMessage=Ping
webSocketRefreshInterval=10
# General settings
httpVersion=1.1
responseTimeout=2
```

You can create base64 string using the unix base64 command:

```
e.g.
echo -n username:password | base64
dXNlcm5hbWU6cGFzc3dvcmQ=
```

This string can then be used in the header setting directly.

```
httpHeaders=Authorization=Basic dXNlcm5hbWU6cGFzc3dvcmQ=,Content-Type=application/json
httpVersion=1.1
```

Initialisation log messages:

```
WebSocket module initialization complete.
ResynchLatch count down by 1 Resync Latch Current Count: 0
TokenLatch waiting for latch to be released
TokenLatch timed out waiting for latch.
TokenLatch latch released.
Websocket connecting to localhost:8080
WebSocket URI: ws://localhost:8080/
Web socket channel is active.
Probe websocket connection has been established
WebSocket client connected to localhost:8080
Websocket channel active.
Probe connected
```

Ping refresh sending:

```
Inserting data into websocket Ping frame: Ping
Sending websocket ping frame.
Ping frame sent.
```

## 3.2 Ciena MCP Example

File : message_bus_ciena_mcp.props

```
Host                        : 'developer.ciena.com'
Port                        : 443
Username                    : 'guest'
Password                    : 'password'
EnableSSL                   : 'true'
KeyStore                    : '$OMNIHOME/probes/linux2x86/websocket_cienamcp.jks'
KeyStorePassword            : 'netcool'

TransportType               : 'WebSocket'
TransportFile               : '$OMNIHOME/probes/linux2x86/cienaMcpTransport.properties'
TransformerFile             : '$OMNIHOME/probes/linux2x86/ciena_mcp_parser_config.json'
MessageLog                  : '$OMNIHOME/log/message_bus_ciena_mcp.log'
RulesFile                   : '$OMNIHOME/probes/linux2x86/message_bus_ciena_mcp.rules'

InitialResync               : 'true'
HeartbeatInterval           : 60
# EOF
```

File : cienaMcpTransport.properties

```
httpVersion=1.1
httpHeaders=Accept=application/json,User-Agent=IBM Netcool/OMNIBus Message Bus Probe
responseTimeout=60
securityProtocol=TLSv1.2

keepTokens=token,user,RSV_TOTAL:/meta/query-total

loginRequestURI=/tron/api/v1/tokens
loginRequestMethod=POST
loginRequestContent={\"username\":\"++Username++\",\"password\":\"++Password++\"}


resyncRequestURI=/nsa/api/v2_0/alarms/filter/filteredAlarms?filter%5Bstate%5D%5B
%5D=ACTIVE&filter%5Bseverity%5D%5B%5D=CRITICAL%2CMAJOR%2CMINOR%2CWARNING&offset=+
+{0,100,100}++&pageSize=100
resyncRequestMethod=GET
resyncRequestHeaders=Authorization=Bearer ++token++

loginRefreshURI=/tron/api/v1/tokens
loginRefreshMethod=POST
loginRefreshContent={\"username\":\"++Username++\",\"password\":\"++Password++\"}
loginRefreshInterval=3600
refreshRetryCount=5

webSocketURI=/kafkacomet/socket/websocket?user_id=++user++&vsn=1.0.0
webSocketHeaders=Authorization=Bearer ++token++
webSocketSubscribeMessage={\"topic\":\"topics:bp.aeprocessor.v1.alarms\",\"ref\":0,\"even
t\":\"phx_join\",\"payload\":{}}
webSocketRefreshMessage={\"topic\":\"topics:bp.aeprocessor.v1.alarms\",\"event\":\"heartb
eat\",\"payload\":{},\"ref\":\"1\"}
webSocketRefreshInterval=20
```

Creating the KeyStore file

The server certificate can be downloaded from the server using openssl.

```
./get_ssl.sh developer.ciena.com 443

keytool -printcert -file ./server.cert

keytool -import -keystore websocket_cienamcp.jks -alias developer.ciena.com -file
server.cert -storepass netcool
```

Download the certificates CA.cert from the CA web site [Entrust Certification Authority - L1K]

```
keytool -printcert -file ./CA.cert

keytool -import -keystore websocket_cienamcp.jks -alias CA -file CA.cert -storepass
netcool

keytool -list -keystore websocket_cienamcp.jks -storepass netcool
```

### 3.2.1 get_ssl.sh

File : get_ssl.sh

```
#! /bin/sh
if [ $# != 2 ]
then

echo "Usage : `basename $0` [host] [port] "
exit
fi

host=$1
port=$2

echo -n | openssl s_client -connect ${host}:${port}  | sed -ne '/-BEGIN CERTIFICATE-/,/-
END CERTIFICATE-/p' > server.cert
#EOF
```

# 4 Webhook Transport

The Message bus probe comes with an example Webhook transport properties file:
Direcotry : $NCHOME/omnibus/java/conf/default
File : **restWebhookTransport.properties**

The Webhook part of the integration is:

```
####
# Webhook Properties
#
####
# The webhookURI property specifies the path of the webhook on
# the local server. If only the path is specified, the HTTP server
# will bind to the local port number specified in the Port probe property.
# The specified local port must be a free port.
# This property can be set to a URL to specify the scheme (http
# or https), port number, and path. If unset or empty, a unique path
# will be generated during startup and printed in the probe log at INFO log level.
# Example: webhookURI=/probe/webhook or webhookURI=http://hostname:80/probe/webhook
# Default is OFF (HTTP server disabled)
webhookURI=/


#respondWithContent=OFF
#validateBodySyntax=ON
#validateRequestURI=ON

# If set to a positive integer value, it sets the timeout parameter (in seconds) that
# the webhook will allow an idle connection to remain open
# before it is closed. If set to 0 , not timeout is applied and the HTTP server
# will keep all connections.
# Default : 180
#idleTimeout=180
#
# Webhook (Server) Basic Authentication. Clients must use Basic Authentication
# with the correct credential
# To enable Basic Authentication, set the serverBasicAuthenticationUsername
# and serverBasicAuthenticationPassword properties below.
#serverBasicAuthenticationUsername=
#serverBasicAuthenticationPassword=
#EOF
```

Top of file settings:

```
# Specifies the HTTP version. Supports version 1.1 (default) or 1.0.
#httpVersion=1.1

# A comma-seperated list of HTTP headers to be set in all
# outgoing HTTP request.
#httpHeaders=

# The timeout (in seconds) to wait for the HTTP response
# from the server. Default is 60 seconds.
#responseTimeout=60

# Set to "ON" to re-establish the connection to the HTTP server
# before sending a HTTP request. Set to "OFF" to disconnect the probe
# after a connection close request is received from the HTTP server.
# Default value is "OFF"
#autoReconnect=OFF

####
# Each HTTP request can be configured by the RequestURI, RequestMethod,
# RequestContent and RequestHeader.
# 1. Specify the path in the RequestURI property. Example : /login
# 2. Specify the HTTP method in the RequestMethod. Example : POST, GET, PATCH, etc.
# 3. (Optional) Specify the HTTP body in the RequestContent. Example: {\"sample\":\"json\"}
# 4. (Optional) Specify the additional HTTP headers in the RequestHeaders property. This
#    will override any headers set in the httpHeader property.
# 5. Specify the interval (in seconds) for loginRefresh and subscribeRefresh request property
#    to enable a HTTP request to be sent periodically.
####
# Constructs the HTTP request which will be sent after the OAuth
# access token request.
#loginRequestURI=
#loginRequestMethod=
#loginRequestContent
#loginRequestHeaders=

# Constructs the HTTP request which will be sent periodically
# as specified by the loginRefreshInterval.
#loginRefreshURI=
#loginRefreshMethod=
#loginRefreshContent=
#loginRefreshHeaders=
#loginRefreshInterval=

# Constructs the HTTP request which will be sent before disconnecting.
# It can be used to unsubscribe or logout from the system.
#logoutRequestURI=
#logoutRequestMethod=
#logoutRequestContent=
#logoutRequestHeaders=

# Constructs the HTTP request which will be sent periodically.
# The interval is configured by the ResyncInterval probe property.
#resyncRequestURI=
#resyncRequestMethod=
#resyncRequestContent=
#resyncRequestHeaders=

# Constructs the HTTP request which will be sent in the subscribe state
# after successfully connecting to the target device.
#subscribeRequestURI=
#subscribeRequestMethod=
#subscribeRequestContent=
#subscribeRequestHeaders=

# Constructs the HTTP request which will be sent periodically
# as specified by the subscribeRefreshInterval.
#subscribeRefreshURI=
#subscribeRefreshMethod=
#subscribeRefreshContent=
#subscribeRefreshHeaders=
#subscribeRefreshInterval=
```

```
# A comma-separated list of attributes to extract from the
# JSON body received in a HTTP response.
#keepTokens=

# The number of retries for the refresh requests. Default is 0 (no retries).
#refreshRetryCount=0

# Specifies the SSL protocol. Example : TLSv1.2
#securityProtocol=

####
# OAuth2 Properties
# - The supported grant type is Resource Owner Password Crendentials Grant.
# The Username and Password property will be used as the
# resourceOwnerUsername and resourceOwnerPassword to obtain the
# access token. Please ensure these probe properties are set.
##

# The token endpoint to request for an access token.
# Default is unset (disabled).
# Specify the path on the remote host or the
# the full token endpoint URL to request for an
# access token. For example: tokenEndpointURI=/oauth/token
# or tokenEndpointURI=http://oauthserverhost:80/oauth/token
#tokenEndpointURI=

# Specifies the basic authentication credentials.
# As an alternative for servers that does not support basic authentication,
# use clientId and clientSecret properties to send the client
# credentials in the request-body.
#basicAuthenticationUsername=
#basicAuthenticationPassword=

# Specifies the client credentials to send in the request-body
# as the "client_id" and "client_secret" attributes.
#clientId=
#clientSecret=

# Specifies the access scopes. Accepts a comma-separated list.
# For example: read,write
#scopes=
```

## 4.1 Simple Webhook Example

A simple Webhook listener can be created using just the probe property file and transport properties file,as shown here.

### 4.1.1 Property file

```
Server                          : 'AGG_P'
ServerBackup                    : 'AGG_B'
# Best Practice
NetworkTimeout                  : 15
PollServer                      : 60
# Buffering
Buffering                       : 1
BufferSize                      : 200
FlushBufferInterval             : 9
# Webhook
Host                            : 'localhost'
Port                             : 8060
TransportType                   : 'Webhook'
TransportFile                   :
'$NCHOME/omnibus/probes/linux2x86/messagebus.webhook.example.Transport.properties'
# Default parsing
MessagePayload                  : 'json'
JsonMessageDepth                : 10
TransformerFile                 : ''
# Heartbeating
HeartbeatInterval               : 0
ProbeWatchHeartbeatInterval     : 60
#EOF
```

### 4.1.2 Transport properties file

```
httpVersion=1.1
webhookURI=/
#respondWithContent=OFF
#validateBodySyntax=ON
validateRequestURI=OFF
#EOF
```

### 4.1.3 Send event script

```
File : send_to_uri.sh
#! /bin/sh
if [ $# != 1 ]
then
echo "Usage : `basename $0` [http://host:port/path/to/dir]"
exit
fi
URI=$1
curl -v -H "Content-Type: application/json" -d '{"test":"Message text"}' -X POST ${URI}
#EOF
```

### 4.1.4 Sending an event

Probe is ready:

Log messages

```
Webhook URL:http://<FQDN>:8060/
Probe connected
```

Checking:

```
netstat -na | grep 8060
```

Sending an event:

```
./send_to_uri.sh http://<FQDN>:8060/
```

Log messages

```
Received end of HTTP content
Received message with length of 23 from endpoint /: {"test":"Message text"}

[Event Processor] resync_event:      false
[Event Processor] MESSAGE.META.Host: [<FQDN>:8060]
[Event Processor] MESSAGE.META.Content-Type: [application/json]
[Event Processor] test:      Message text
[Event Processor] MESSAGE.META.User-Agent:   [curl/<VERSION>]
[Event Processor] MESSAGE.META.Content-Length:    [23]
[Event Processor] MESSAGE.META.Sender_address:    [/<IP>:<PORT>]
[Event Processor] MESSAGE.META.Accept:       [*/*]
```

## 4.2 Basic Authentication Example

You can run the Webhook listener with basic authentication, which provides some security and prevents unwanted access of the URI.

### 4.2.1 Property file:

```
Server                          : 'AGG_P'
ServerBackup                    : 'AGG_B'
# Best Practice
NetworkTimeout                  : 15
PollServer                      : 60
# Buffering
Buffering                       : 1
BufferSize                      : 200
FlushBufferInterval             : 9
# Webhook
Host                            : 'localhost'
Port                            : 8070
TransportType                   : 'Webhook'
#
MessagePayload                  : 'json'
JsonMessageDepth                : 10
TransportFile                   :
'$NCHOME/omnibus/probes/linux2x86/messagebus.webhook.basicauth.example.Transport.properti
es'
TransformerFile                 : ''
# Heartbeating
HeartbeatInterval               : 0
ProbeWatchHeartbeatInterval     : 60
#EOF
```

### 4.2.2 Transport Properties file

```
####
# Webhook Properties
####
# The webhookURI property specifies the path of the webhook on
# the local server. If only the path is specified, the HTTP server
# will bind to the local port number specified in the Port probe property.
# The specified local port must be a free port.
# This property can be set to a URL to specify the scheme (http
# or https), port number, and path. If unset or empty, a unique path
# will be generated during startup and printed in the probe log at INFO log level.
# Example: webhookURI=/probe/webhook or webhookURI=http://hostname:80/probe/webhook
# Default is OFF (HTTP server disabled)
webhookURI=/
#respondWithContent=OFF
#validateBodySyntax=ON
validateRequestURI=OFF
idleTimeout=30
#
# Webhook (Server) Basic Authentication. Clients must use Basic Authentication
# with the correct credential
# To enable Basic Authentication, set the serverBasicAuthenticationUsername
# and serverBasicAuthenticationPassword properties below.
serverBasicAuthenticationUsername=username
serverBasicAuthenticationPassword=password
#EOF
```

### 4.2.3 Sending events

Probe start-up log:

```
Webhook URL:http://<FQDN>:8070/
Probe connected
```

Sending an unauthorized message:

```
./send_to_uri.sh http://<FQDN>:8070/
...
< HTTP/1.1 401 Unauthorized
```

Unauthorized [no basic authentication] probe log:

```
Checking client credentials.
Authorization HTTP header not found or unset.
Not adding payload to the response for payload type UNASSIGNED
Sending response 401 Unauthorized
```

Unauthorized [Invalid basic authentication] probe log:

```
Checking client credentials.
Found Authorization HTTP header with basic authentication scheme.
Invalid authentication credential provided in client request.
Done checking client authentication credentials.
Not adding payload to the response for payload type UNASSIGNED
Sending response 401 Unauthorized
```

Defining the username/password in the header:

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

Sending the message using curl:

```
curl -v -H "Content-Type: application/json"
-H "Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ="
-d '{"test":"Message"}' -X POST http://<FQDN>:8070
< HTTP/1.1 200 OK
```

Probe Log:

```
Received end of HTTP content
Checking client credentials.
Found Authorization HTTP header with basic authentication scheme.
Client authentication credential accepted.
...
Received message with length of 18 from endpoint /: {"test":"Message"}
...
resync_event:      false
MESSAGE.META.Host: [<FQDN>:8070]
MESSAGE.META.Authorization:     [Basic dXNlcm5hbWU6cGFzc3dvcmQ=]
MESSAGE.META.Content-Type: [application/json]
test:      Message
MESSAGE.META.User-Agent:   [curl/<VERSION>]
MESSAGE.META.Content-Length:    [18]
MESSAGE.META.Sender_address:    [/0:0:0:0:0:0:0:1:57088]
MESSAGE.META.Accept:       [*/*]
```

## 4.3 SSL Example

The additional security of SSL certificates can provide enough security for the probes listener to be exposed. The probe supports both SSL and Basic Authorization if needed. The probes certificate needs to be a Certificate Authority SSL Certificate. The sender needs to trust the CA, through having the CA certificate in its trustsore and to address the probes Webhook port using the FQDN.

### 4.3.1 Property file:

```
Server                          : 'AGG_P'
ServerBackup                    : 'AGG_B'
NetworkTimeout                  : 15
PollServer                      : 60
Buffering                       : 1
BufferSize                      : 200
FlushBufferInterval             : 9
Host                            : 'localhost'
Port                             : 8080
TransportType                   : 'Webhook'
MessagePayload                  : 'json'
JsonMessageDepth                : 10
TransportFile                   :
'$NCHOME/omnibus/probes/linux2x86/messagebus.webhook.ssl.example.Transport.properties'
TransformerFile                 : ''
HeartbeatInterval               : 0
ProbeWatchHeartbeatInterval     : 60
EnableSSL                       : 'true'
KeyStore                        :
'$NCHOME/omnibus/probes/linux2x86/messagebus.webhook.ssl.example/ProbeKeyStore.jks'
KeyStorePassword                : 'netcool'
```

### 4.3.2 Transport Properties file

```
httpVersion=1.1
securityProtocol=TLSv1.2
webhookURI=/
validateRequestURI=OFF
```

### 4.3.3 Sending Events

Probe start-up log:

```
Supported secure socket protocol(s): [TLSv1, TLSv1.1, TLSv1.2]
Enabled secure socket protocol(s): [TLSv1.2]
...
Starting a HTTPS server on local port 8080
Webhook URL:https://<FQDN>:8080/
Probe connected
```

Working : Using the <FQDN>  CURL Command:

CURL Command:

```
curl --cacert ./caroot.cer -v -H "Content-Type: application/json" -d
'{"test":"Message text"}' -X POST https://<FQDN>:8080/
…
< HTTP/1.1 200 OK
...
```

Probe log:

```
Remote client connected: /...,total connected clients: 1
Adding IdleTimeoutHandler to disconnect idle clients after 180 seconds.
SSL Handshake completed successfully.
...
Performing JSON format validation
Checking JSON syntax.
JSON format OK.
Responding OK
Sending response 200 OK
Setting Connection header:keep-alive: timeout=180
No HTTP body. Content-Length header set to 0.
response.status().code(): 200
Received message with length of 23 from endpoint /: {"test":"Message text"}
Start parsing JSON message
Adding event of size 2 to queue
Keep the connection.
Response successfully sent.
Done parsing JSON message
Connection close request received from remote client. Connection closed.
Remote client disconnected: /...,total connected clients: 0
[Event Processor] resync_event:      false
[Event Processor] MESSAGE.META.Host: [<FQDN>:8080]
[Event Processor] MESSAGE.META.Content-Type: [application/json]
[Event Processor] test:      Message text
[Event Processor] MESSAGE.META.User-Agent:   [curl/7.29.0]
[Event Processor] MESSAGE.META.Content-Length:    [23]
[Event Processor] MESSAGE.META.Sender_address:    [/...]
[Event Processor] MESSAGE.META.Accept:       [*/*]
```

Broken : Using the hostname CURL Command:

CURL Command:

```
curl --cacert ./caroot.cer -v -H "Content-Type: application/json" -d '{"test":"Message
text"}' -X POST https://<hostname>:8080/

* About to connect() to <hostname> port 8080 (#0)
*   Trying ...
* Connected to <hostname> (IP) port 8080 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
*   CAfile: ./caroot.cer
  CApath: none
* Server certificate:
*       subject: CN=<FQDN>,OU=Netcool,O=IBM,L=New York,ST=New York,C=S{...
*       start date: ... GMT
*       expire date: ... GMT
*       common name: <FQDN>
      issuer: E=root@<FQDN>,CN=<FQDN>,OU=Netcool,O=IBM,L=New York,ST=New York,C=US
      NSS error -12276 (SSL_ERROR_BAD_CERT_DOMAIN)
```

Probe log:

```
Remote client connected: /...,total connected clients: 1
Adding IdleTimeoutHandler to disconnect idle clients after 180 seconds.
SSL Handshake unsuccessful.
Connection close request received from remote client. Connection closed.
HttpServerRequestHandler caught an exception.
Exception: javax.net.ssl.SSLException: Received fatal alert: bad_certificate
Disconnecting webhook client: /...
Webhook Subscription flag set to false
Remote client disconnected: /...,total connected clients: 0
```

## 4.4 Secure Webhook – HTTPS and Basic Authentication

The maximum security available to the message bus probe, is a HTTPS Webhook listener using basic authentication. Review the earlier chapters to see the details on how to configure a HTTPS Webhook listener and basic authentication separately.

### 4.4.1 Property file

```
# Define a unique instance and rules
Name                 : 'message_bus_webhookbasicauth_ssl'
RulesFile            :
'$NCHOME/omnibus/probes/linux2x86/mb_webhookbasicauth_ssl/discard.rules'
###
# User and Password
AuthUserName         : 'probe'
AuthPassword         : 'netcool'

Server               : 'AGG_P'
ServerBackup         : 'AGG_B'
###
# Transport settings
TransportType        : 'Webhook'
TransportFile        :
'$NCHOME/omnibus/probes/linux2x86/mb_webhookbasicauth_ssl/Transport.properties'
Host                 : "0.0.0.0"
Port                 : 8443
# ++Username++ and ++Password++ does not work for basic authentication
HeartbeatInterval    : 0
MessageLevel         : 'info'
MessageLog           : '$NCHOME/omnibus/log/message_bus_webhook_basic_ssl.log'
# Basic JSON parsing
JsonNestedPayload    : 'json'
JsonMessageDepth     : 10
# JSON default parser
TransformerFile      : ''
# SSL
EnableSSL            : 'true'
KeyStore             : '$NCHOME/omnibus/utils/SSLWEBHOOK_ROOT_CA/CACERTS/keystore.jks'
KeyStorePassword     : 'netcool'
###
# Best practice
NetworkTimeout       : 15
PollServer           : 60
# Buffering for 200 events per second
Buffering            : 1
BufferSize           : 2000
FlushBufferInterval  : 11
# Tuning
MaxEventQueueSize    : 50000
TransportQueueSize   : 50000
DisableDetails       : 1
```

### 4.4.2 Transport properties

In the transport properties file the webhookURI can be set to '/' with validateRequestURI set to 'OFF' to allow any path to be used [seen as endpoint in the probes logging]. The transport file does allow ++Host++ and ++Port++ to be used, but does not allow ++Username++ and ++Password++ in the basic authentication settings.
With the latest test fix patch the allowPOSTOnly can be used to disable unwanted methods being processed by the probe [Process POST requests only].

```
webhookURI=https://++Host++:++Port++/
respondWithContent=OFF_WITH_HTTP_200
validateRequestURI=OFF
idleTimeout=20
#
# Obtaining the basic auth string
# echo -n USERNAME:PASSWORD | base64
# Returns : VVNFUk5BTUU6UEFTU1dPUkQ=
#
serverBasicAuthenticationUsername=USERNAME
serverBasicAuthenticationPassword=PASSWORD
# Prevent port scanners
allowPOSTOnly=ON
# EOF
```

### 4.4.3 Property and properties password encryption

To further secure the probes configuration, the password strings used in the message bus probe configuration can be encrypted:

The property and transport properties file password are encrypted using an AES_FIPS encryption key file.

Create the key file used by the probe[s] (only needs to be run once]).

```
mkdir $NCHOME/etc/security/keys_files
$NCHOME/omnibus/bin/nco_keygen -l 256 -o $NCHOME/etc/security/keys_files/mb_probe.keyfile
```

Encrypt the password property and properties as required.

```
$OMNIHOME/bin/nco_aes_crypt -c AES_FIPS -k $NCHOME/etc/security/keys_files/mb_probe.keyfile "netcool" -o
/tmp/encrypted_AuthPassword.txt

$OMNIHOME/bin/nco_aes_crypt -c AES_FIPS -k $NCHOME/etc/security/keys_files/mb_probe.keyfile "netcool" -o
/tmp/encrypted_KeyStorePassword.txt

$OMNIHOME/bin/nco_aes_crypt -c AES_FIPS -k $NCHOME/etc/security/keys_files/mb_probe.keyfile "netcool" -o
/tmp/encrypted_serverBasicAuthenticationPassword.txt
```

With the '-o filename.txt' option, the string is written to a file; Which is useful to ensure the string is copied completely.
Remember to remove the temporary files, once the encrypted string is saved.

Add the required property settings for AES_FIPS to the probe property file.

```
# Cryptology property settings
ConfigCryptoAlg : 'AES_FIPS'
ConfigKeyFile : '$NCHOME/etc/security/keys_files/mb_probe.keyfile'
```

Add in the text from the encrypted text file to the probe property and transport properties files.

For example.

Probe property file.

```
# Object Server User and Password
AuthUserName : 'probe'
AuthPassword : '@44:INExeHJFVpZrBp22G0B8jO5czTQ+/3hOSqUjcuDDJv8=@'

# SSL Key store
EnableSSL            : 'true'
KeyStore             : '$NCHOME/omnibus/utils/SSLWEBHOOK_ROOT_CA/CACERTS/keystore.jks'
KeyStorePassword     : '@44:QQO/qj7sKTXC3IeuCFpE74HBXfrtS1I0oE4DH+qO5XY=@'
```

Transport properties file.

```
# Basic authentication username and password
serverBasicAuthenticationUsername=USERNAME
serverBasicAuthenticationPassword=@44:EdXRlu1plepM81zjyZWnEYSppNBzsSEN8t/iDgVkzew=@
```

## 4.5 Custom HMAC Signature security

The Message bus probe can be configured to provide HMAC security using the pre-parser Java Script feature and rules file logic to compare the Header HMAC signature with the pre-parser Java Script's signature. Provided the encodings both use the same passphrase the two signature strings will match for the same JSON data.
The ability to process the events is defined by the probes rules file logic, rather than any hard coded security.

### 4.5.1 Probe property file

```
Name                          : 'message_bus_webhook_http_hmac'
RulesFile                     : '$NCHOME/omnibus/probes/linux2x86/signature.rules'

TransportType                 : 'Webhook'
TransportFile                 : '$NCHOME/omnibus/probes/linux2x86/WebhookTransportHTTP8080_HMAC.properties'

MessagePayload                : 'json'
JsonMessageDepth              : 10
TransformerFile               : '$NCHOME/omnibus/probes/linux2x86/message_bus_parser_hmac.json'
```

### 4.5.2 Transport properties file

File : WebhookTransportHTTP8080_HMAC.properties

```
webhookURI=http://0.0.0.0:8080/
validateRequestURI=OFF
idleTimeout=19
```

### 4.5.3 Probe rules file

The rules file logic checks and compares the two signatures to make sure they are the same; Discarding any events where there is no match.

File : signature.rules

```
if (exists($(MESSAGE.META.X-Webhook-Signature))) {
  if (exists($SIGNATURE)) {

    if (regmatch($(MESSAGE.META.X-Webhook-Signature),"\[(.*)\]") ) {
      $WEBHOOK_SIGNATURE = extract ($(MESSAGE.META.X-Webhook-Signature),"\[(.*)\]")
    } else {
      log(DEBUG,"Unexpected format - MESSAGE.META.X-Webhook-Signature - missing brackets")
    }
# Debuglogging
# log(DEBUG,"MESSAGE.META.X-Webhook-Signature = >" + $(MESSAGE.META.X-Webhook-Signature) + "<"  )
# log(DEBUG,"WEBHOOK_SIGNATURE = >" + $WEBHOOK_SIGNATURE + "<" )
# log(DEBUG,"SIGNATURE = >" + $SIGNATURE + "<" )

    if (match($WEBHOOK_SIGNATURE,$SIGNATURE)) {
      log(DEBUG,"PROCESSING : " + $*)
    } else {
      log(DEBUG,"DISCARDING : INVALID MESSAGE.META.X-Webhook-Signature : " + $*)
      discard
    }

  } else {
    log(DEBUG,"DISCARDING: No Signature in data [check java script pre-parser]" + $*)
    discard
  }
} else {
  log(DEBUG,"DISCARDING: No Signature in header" + $*)
  discard
}
```

### 4.5.4 Probe parser file

File : message_bus_parser_hmac.json

```
{
  "eventSources" : [ {
    "endpoint" : "/resync",
    "name" : "ResyncAlarmParser",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10,
      "headerPrefix" : "",
      "preparserScript" :
"/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/verify.js"
    }
  }, {
    "name" : "OtherAlarmParser",
    "type" : "ANY",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10,
      "headerPrefix" : "",
      "preparserScript" :
"/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/verify.js"
    }
  }]
}
```

### 4.5.5 Pre-parser Java Script

The pre-parser Java Script defines the passphrase, defined by the 'secret' variable, and hard coded in the script. The pre-parser Java Script inserts the signature into the original JSON data for comparison in the probes rules file logic.

File : verify.js

```
function preParser(rawPayload, endpoint) {
    var secret = "your_shared_secret_here";
    var enrichedJson = {};

    try {
        if (!rawPayload) {
            return rawPayload;
        }

        // 1. Convert input to standard Java/JS String
        var payloadStr = (typeof rawPayload === 'string')
            ? rawPayload
            : new java.lang.String(rawPayload, "UTF-8");

        // 2. Compute HMAC-SHA256 using standard Java Cryptography Architecture (JCA)
        var Mac = Java.type("javax.crypto.Mac");
        var SecretKeySpec = Java.type("javax.crypto.spec.SecretKeySpec");

        var mac = Mac.getInstance("HmacSHA256");
        var secretKeySpec = new SecretKeySpec(
            new java.lang.String(secret).getBytes("UTF-8"),
            "HmacSHA256"
        );
        mac.init(secretKeySpec);

        var hmacBytes = mac.doFinal(
            new java.lang.String(payloadStr).getBytes("UTF-8")
        );

        // 3. Convert HMAC bytes to hex string
        var hexChars = "0123456789abcdef";
        var computedSignatureHex = "";
        for (var i = 0; i < hmacBytes.length; i++) {
            var b = hmacBytes[i] & 0xFF;
            computedSignatureHex += hexChars.charAt(b >>> 4);
            computedSignatureHex += hexChars.charAt(b & 0x0F);
        }

        // 4. Parse the original JSON payload
        var originalJson = JSON.parse(payloadStr);

        // 5. Add the computed signature to the JSON object
        enrichedJson = originalJson;
        enrichedJson["SIGNATURE"] = computedSignatureHex;

        // Return the modified JSON string or object as required by the parser
        return JSON.stringify(enrichedJson);

    } catch (e) {
        // Log error if logger is available, or return original payload
        return rawPayload;
    }
}
```

### 4.5.6 HMAC signature test script

For the clients JSON data to be processed by the rules file logic, the client needs to send the correct signature in the header.

```
-H "X-Webhook-Signature: $SIGNATURE" \
```

The passphrase is used to create the signature, along with the JSON data.

```
-hmac "your_shared_secret_here"
```

File : send_and_encode_file_to_uri.sh

```
#! /bin/sh
export HOST PORT URI FILE

# Parse input parameters

if [ $# -ne 4 ]
then
echo "Usage : `basename $0` [host] [port] [uri] [file]"
exit
else
HOST=$1
PORT=$2
URI=$3
FILE=$4
fi

# Create SIGNATURE from passphrase and incoming json data
SIGNATURE=$(openssl dgst -sha256 -hmac "your_shared_secret_here" -hex < ${FILE} | sed
's/^.*= //')

echo "SIGNATURE=$SIGNATURE"

# Send json data

curl -v \
  -H "Content-Type: application/json" \
  -H "X-Webhook-Signature: $SIGNATURE" \
  --data-binary @${FILE} \
  -X POST http://${HOST}:${PORT}${URI}

#EOF
```

File : test_message.json

```
{"Message":"Test"}
```

## 4.6 Webhook OAuth example from the manual

The OAuth 2.0 authentication is provided by the service the probe connects to.

### 4.6.1 Probe Property file

File : message_bus_oauth.props

```
# SSL connectivity
EnableSSL                    : 'true'
KeyStore                     : '$NCHOME/omnibus/probes/linux2x86/oauth_ssl/ProbeStore.jks'
KeyStorePassword             : 'netcool'
# Transport
TransportType                : 'Webhook'
TransportFile                : '$NCHOME/omnibus/probes/linux2x86/oauthTransport.properties'
# OAUTH server
Host                         : '<OAUTH-SERVER>'
Port                         : 443
Username                     : '<OAUTH-USER>'
Password                     : '<OAUTH-PASSWORD>'
# JSON
TransformerFile              : ''
MessagePayload               : 'json'
# Heartbeating
ProbeWatchHeartbeatInterval  : 0
HeartbeatInterval             : 60
```

### 4.6.2 Transport Properties

File : oauthTransport.properties

```
# General
httpVersion=1.1
responseTimeout=60
securityProtocol=TLSv1.2
httpHeaders=Accept=application/json,Content-Type=application/json
# httpHeaders=Authorization=Bearer ++Oauth.access_token++
# httpHeaders=Authorization=Basic ++Username++:++Password+
+,Accept=application/json,Content-Type=application/json

# Basic Authentication
basicAuthenticationUsername=++Username++
basicAuthenticationPassword=++Password++
# Secret
clientId=
clientSecret=
scope=

# OAuth ++Oauth.access_token++
tokenEndpointURI=https://++Host++:++Port++/api/v1/oauth/tokens

# Subscribe
subscribeRequestURI=/api/v1/alarms
subscribeRequestMethod=POST
subscribeRequestContent={"callback": "++WebhookUrl++", "fields": "*"}
subscribeRequestHeaders=Accept=application/json,Content-
Type=application/json,Authorization=Bearer ++Oauth.access_token++

# Webhook ++WebhookUrl++
webhookURI=http://<probe-server>:8888/ciena/webhook
#EOF
```

## 4.7 Cisco EPNM Custom example

This example outlines a possible configuration for the CISCO EPNM server which supports:

- login
- subscribe
- Webhook
- basic authentication
- SSL
- JSON

Note: There is a formal integration for Cisco EPNM using the Message bus integration.

### 4.7.1 Probe property

```
# Webhook transport and json parsing
TransportType       : 'Webhook'
TransportFile       : '${NCHOME}/omnibus/probes/linux2x86/ciscoepnmTransport.properties'
MessagePayload      : 'json'
JsonMessageDepth    : 10
InitialResync       : 'false'
# Cisco epnm server
Host                : 'epnm_host'
Port                : epnm_port
# Username & password
Username            : 'username'
Password            : 'password'
# SSL configuration
EnableSSL           : 'true'
KeyStore            : '${NCHOME}/omnibus/probes/linux2x86/epnm_ssl/probestore.jks'
KeyStorePassword    : 'netcool'
# EOF
```

### 4.7.2 Transport properties:

```
# HTTP settings
httpVersion=1.1
securityProtocol=TLSv1.2
responseTimeout=10
# Always use basic authentication
httpHeaders=Authorization=Basic ++Username++:++Password++
# Keep session token
keepTokens=JSESSIONID
# Basic authentication
basicAuthenticationUsername=++Username++
basicAuthenticationPassword=++Password++
# login to get tokens
loginRequestURI=/restconf/data/v1/cisco-customer:customer
loginRequestMethod=GET
loginRequestHeaders=Authorization=Basic ++Username++:++Password++, accept:
application/json
# Subscription request to send to probes Webhook listener
subscribeRequestURI=/restconf/data/v1/cisco-notifications:subscription
subscribeRequestMethod=POST
subscribeRequestContent={\"push.endpoint-
url\":\"https://<probe_ip>:8080/probe/webhook\",\"push.topic\":\"<topic>\",\"push.format\
": \"json\"}
subscribeRequestHeaders==Authorization=Basic ++Username++:++Password++, accept:
application/json
# Create Webhook listener
webhookURI=https://<probe_ip>:8080/probe/webhook
# Comments
# <topic> can be either
# "inventory", "service-activation", “alarm”, “template- execution” or "ha"
#EOF
```

## 4.8 Cisco APIC probe

The current Cisco APIC probe cannot use the latest transport module which includes the log4j security fixes.

```
PATCH common-transportmodule-34
REVISION 0
  - Updated dependency libraries (cherry-picked from hotfix-v33.x):
      - Log4j upgraded to 2.16.0 to resolve Log4Shell CVE-2021-44228 and CVE-2021-45046.
      - Log4j upgraded to 2.17.1 to resolve Log4Shell CVE-2021-44832.
```

The Message Bus probe release 19.0 and above can connect to the Cisco APIC v5.2 and above.

```
Version output details.
nco_p_message_bus -version
Version 1.8.0
Release ID: 19.0.10
Transformer Release ID: 11.0.5
TransportModule Release ID: 34.0.9
```

The message bus probe can use the Cisco APIC probes property and transport properties file.
For the message bus probe property settings, the WebSocketID setting is commented out, since it is not supported by the Message Bus probe.

```
#WebSocketID : 'subscriptionId'
```

**Note:**
For transport settings, the defunct WebSocketID value (subscriptionId) is defined in the transport properties file in the keepTokens attribute.

```
#### - subscriptionId set in keepTokens for Cisco APIC ####
keepTokens=subscriptionId
```

For the newer versions of Cisco APIC there is greater security. Ensure that the user defined in the probe property file has permission to subscribe and resynchronise events.

The Cisco APIC requires the Cisco-APIC cookie to perform commands such as subscribing and resynchronisation. The curl command supports the storing and sending of cookies using a text file.

```
curl --cookie cookie.txt --cookie-jar cookie.txt <url>
```

Example curl commands.

Login.

```
curl -k -v -H "Content-Type: application/json" \
--cookie cookies.txt --cookie-jar cookies.txt \
-d '{"aaaUser":{"attributes":{"name":"CISCO-USER","pwd":"CISCO-PASSWORD"}}}'
-X POST http://CISCO-HOST:8080/api/aaaLogin.json
```

Resynchronisation.

```
curl -k -v -H "Content-Type: application/json" \
--cookie cookies.txt --cookie-jar cookies.txt \
-X GET http://CISCO-HOST:8080/api/class/faultInst.json
```

### 4.8.1 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>/CiscoAPIC.Transport.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'
# *** Cisco APIC
Cookie                          : 'APIC-cookie'
# Event parsing
MessagePayload                  : 'json.imdata.faultInst.attributes'
JsonMessageDepth                : 10
# EOF
```

### 4.8.2 CiscoAPIC.transport.properties

```
# Login/Logout
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++\"}}}"


# Resynchroisation

resyncRequestURI=/api/class/faultInst.json
resyncRequestMethod=GET
resyncRequestContent=

# Subscription to websocket
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

# Settings

httpVersion=1.1
securityProtocol=TLSv1.2


#### - subscriptionId set in keepTokens for Cisco APIC ####
keepTokens=subscriptionId
responseTimeout=10

# EOF
```

## 4.9 Resynchronisation example

The Message bus probe can be configured to poll a URI periodically using the resynchronisation feature, and suitable rules file logic. If the synchronisation URI accepts a filter this can be used to limit the events pulled from the server.

### 4.9.1 Probe property file

```
Host            : 'localhost'
Port            :  8888
TransportType   : 'Webhook'
TransportFile   : '${OMNIHOME}/probes/linux2x86/resynch-webhookTransport.properties'
# Resynchronisation settings
InitialResync   : 'true'
ResyncInterval  : 60
RulesFile       : '$OMNIHOME/probes/linux2x86/message_bus-resynch.rules'
# Heartbeating on or off [0]
HeartbeatInterval : 0
#EOF
```

### 4.9.2 Transport properties file

```
# HTTP and auto-reconnect
httpVersion=1.1
autoReconnect=ON
# Example Resynchronisation URI
resyncRequestURI=/probe/events
resyncRequestMethod=GET
# Webhook settings
webhookURI=http://localhost:7777/
validateRequestURI=OFF
idleTimeout=180
# EOF
```

### 4.9.3 Rules file logic

```
# Arrays defined at the top of rules files
# Array to store event times
array   EventTimeLookup


# Determine $EVENT_TIME exists and create UNIX time token
if ( exists($EVENT_TIME))
{
  $EventTime = DateToTime($EVENT_TIME,"%m/%d/%y %T %p")
  $EventIdentifier = $NOTIFICATIONID
  log (DEBUG, "DEBUG: EventTime=" + $EventTime + "(" + $EventIdentifier + ")" )
}
# Store value if not already stored using $EventIdentifier]
if ( match("",EventTimeLookup[$EventIdentifier]) )
{
  EventTimeLookup[$EventIdentifier] = $EventTime
  log (DEBUG, "DEBUG:Stored NEW event = " + $EventIdentifier)
}
##########################################################
# If it's an old event, compare times and discard older events
##########################################################
else
{
  $EventTimeLookup = EventTimeLookup[$EventIdentifier]
  if ( int($EventTime) > int($EventTimeLookup) )
  {
   log (DEBUG, "DEBUG:NEW: EVENT_TIME = " + $EVENT_TIME + " : " + $EventTime )
# Store event time and process
   EventTimeLookup[$EventIdentifier] = $EventTime
 }
 else
 {
log (DEBUG, "DEBUG:OLD: Discarding " + $EventTime + " <= " + $EventTimeLookup)
# Discard old event
   discard
   $discard_event = "true"
 }
}
…
# Main rules file logic processing only for events that were not discarded
if (!match($discard_event,"true"))
{
...
}
```

## 4.10 Webhook with multiple parsing example

The Webhook listener can be created using '/' and parse events based on the endpoint events are sent to.

### 4.10.1 Property file

```
Server                          : 'AGG_P'
ServerBackup                    : 'AGG_B'
# Best Practice
NetworkTimeout                  : 15
PollServer                      : 60
# Buffering
Buffering                       : 1
BufferSize                      : 200
FlushBufferInterval             : 9
# Webhook
Host                            : 'localhost'
Port                             : 8888
TransportType                   : 'Webhook'
TransportFile                   :
'$NCHOME/omnibus/probes/linux2x86/messagebus.webhook.mulltiparser.transport.properties'
#
MessagePayload                  : 'json'
JsonMessageDepth                : 10
TransformerFile                 : '$NCHOME/omnibus/probes/linux2x86/mulltiparser.json'
# Heartbeating
HeartbeatInterval               : 0
ProbeWatchHeartbeatInterval     : 60
# EOF
```

### 4.10.2 Transport properties file

File : messagebus.webhook.mulltiparser.transport.properties

```
httpVersion=1.1
webhookURI=/
validateRequestURI=OFF
# EOF
```

### 4.10.3 Parser file

File : mulltiparser.json

```
{
  "eventSources" : [ {
    "endpoint" : "/probe/webhook",
    "name" : "Webhook",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10
    }
  }, {
    "endpoint" : "/probe/webhook/test1",
    "name" : "Test1",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10
    }
  }, {
    "endpoint" : "/probe/webhook/test2",
    "name" : "Test2",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10
    }
  }, {
    "endpoint" : "/probe/webhook/test3",
    "name" : "Test3",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10
    }
  }, {
    "name" : "GenericParser",
    "type" : "ANY",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10
    }
  }]
}
```

### 4.10.4 Send an event to a URI script

```
File : send_to_uri.sh
#! /bin/sh
if [ $# != 1 ]
then
echo "Usage : `basename $0` [http://host:port/path/to/dir]"
exit
fi
URI=$1
curl -v -H "Content-Type: application/json" -d '{"test":"Message text"}' -X POST ${URI}
# EOF
```

### 4.10.5 Example Results

```
./send_to_uri.sh http://localhost:8888/probe/webhook

Parser configuration for endpoint /probe/webhook found.


./send_to_uri.sh http://localhost:8888/probe/webhook/test1

Parser configuration for endpoint /probe/webhook/test1 found.


./send_to_uri.sh http://localhost:8888/probe/webhook/test2

Parser configuration for endpoint /probe/webhook/test2 found.


./send_to_uri.sh http://localhost:8888/probe/webhook/test3

Parser configuration for endpoint /probe/webhook/test3 found.


./send_to_uri.sh http://localhost:8888/probe/webhook/test4

No exact parser for endpoint /probe/webhook/test4 found.
Parser configuration for endpoint /probe/webhook will be used.
```

### 4.10.6 Sending an event

When the probe is ready:

Information log messages

```
Webhook URL:http://<FQDN>:8060/
Probe connected
```

Checking the webhook is ready using netstat:

```
netstat -na | grep 8060
```

Sending a test event:

```
./send_to_uri.sh http://<FQDN>:8060/
```

Log messages

```
Received end of HTTP content
Received message with length of 23 from endpoint /: {"test":"Message text"}

[Event Processor] resync_event:      false
[Event Processor] MESSAGE.META.Host: [<FQDN>:8060]
[Event Processor] MESSAGE.META.Content-Type: [application/json]
[Event Processor] test:      Message text
[Event Processor] MESSAGE.META.User-Agent:   [curl/<VERSION>]
[Event Processor] MESSAGE.META.Content-Length:    [23]
[Event Processor] MESSAGE.META.Sender_address:    [/<IP>:<PORT>]
[Event Processor] MESSAGE.META.Accept:       [*/*]
[Event Processor] MESSAGE.META.MSG_ENDPOINT:       /
```

### 4.10.7 endpoint.rules

The newer message bus probe creates the MESSAGE.META.MSG_ENDPOINT token which can be used to process endpoint events differently.

```
    switch($(MESSAGE.META.MSG_ENDPOINT)){
        case "/splunk":
           log(info,"DEBUG: /splunk rules")
           include "splunk.rules"
        case "/webhook/endpoint":
            log(info,"DEBUG: /webhook/endpoint rules")
            include "endpoint.rules"
        case "/notfications":
            log(info,"DEBUG: /notfications rules")
            include "notifications.rules"
        case "/alarms":
            log(info,"DEBUG: /alarms rules")
            include "alarm.rules"
        default:
            log(info,"DEBUG: Default rules")
            include "webhook.rules"
        }
```

## 4.11 Webhook and exploding event array example

The following example is provided as an example on how to parse events using the default parser, and generate unique events from token arrays using genevent. This is sometimes required when there are multiple event arrays in the JSON data, or there is some problem parsing the JSON data, using the JSON parser file.

### 4.11.1 Property File

```
Server          : 'AGG_P'
NetworkTimeout : 15
PollServer      : 60
# Transport settings
TransportType   : 'Webhook'
TransportFile   : '$NCHOME/omnibus/probes/linux2x86/testing-webhook-
Transport.properties'
Host            : 'localhost'
Port            : 11111
# Default parsing
TransformerFile    : ''
MessagePayload     : 'json'
JsonMessageDepth   : 10
HeartbeatInterval : 0
```

### 4.11.2 Transport properties

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

### 4.11.3 Testing

The message bus probe can be run from the command line when testing rules file logic, with events sent the probes port using the curl command. In this example, a script is used to send a JSON event file.

Run the probe from the command line using the test property and properties file, referencing the example explode_instances.rules file.

```
$NCHOME/omnibus/probes/nco_p_message_bus -rulesfile ./explode_instances.rules -propsfile
message_bus_webhook_testing.props -messagelog stdout -messagelevel debug

Then send the event file.

./send_file_to_uri.sh localhost 11111 / ./testing_nested.json
```

JSON Event File : testing_nested.json

```
{"instances":[{"Id":"node001","Name":"TEST001","Location":"Here"},
{"Id":"node002","Name":"TEST002","Location":"There"}]}
```

Example event tokenisation:

```
[Event Processor] resync_event:      false
[Event Processor] instances.0.Id:    node001
[Event Processor] instances.0.Name:  TEST001
[Event Processor] instances.0.Location:      Here
[Event Processor] instances.1.Id:    node002
[Event Processor] instances.1.Name:  TEST002
[Event Processor] instances.1.Location:      There
```

Array debug logging:

```
DEBUG: instances_Id[0] = node001
DEBUG: instances_Name[0] = TEST001
DEBUG: instances_Location[0] = Here
DEBUG: instances_Id[1] = node002
DEBUG: instances_Name[1] = TEST002
DEBUG: instances_Location[1] = There
```

### 4.11.4 send_file_to_uri.sh script

File : send_file_to_uri.sh

```
#! /bin/sh
# Description:
# Send a file to a given URL : http://${HOST}:${PORT}${URI}
#
# Usage : send_file_to_uri.sh [host] [port] [uri] [file]
#
export HOST PORT URI

if [ $# -ne 4 ]
then
echo "Usage : `basename $0` [host] [port] [uri] [file]"
exit
else
HOST=$1
PORT=$2
URI=$3
FILE=$4
fi
# Send event file using CURL
curl -v -H "Content-Type: application/json" --data @./${FILE} -X POST http://${HOST}:$
{PORT}${URI}
# EOF
```

### 4.11.5 Explode tokens to arrays rules

File : explode_instances.rules

```
# arrays defined at the top
array instances_Id
array instances_Name
array instances_Location


# Initialise variables
$counter = 0
$max_instance = 0

clear( instances_Id )
clear( instances_Name )
clear( instances_Location )

# For each token

foreach ( e in $* )
{
# Check the token
  if(regmatch(e,"^instances\.[0-9]+\..*"))
  {
    $INSTANCE_NUMBER = extract(e,"^instances\.([0-9]+)\..*")
    $TOKEN       = extract(e,"^instances\.([0-9]+)\.(.*)")

    log(DEBUG,"DEBUG: TOKEN = " + $TOKEN )
    log(DEBUG,"DEBUG: INSTANCE_NUMBER= " + $INSTANCE_NUMBER )

    if ( int($INSTANCE_NUMBER) > int($max_instance) )
    {
      $max_instance = int($INSTANCE_NUMBER)
    }


# instances.#.Id
   if(regmatch((e),"^instances\.[0-9]+\.Id"))
   {
     log(DEBUG,"DEBUG: $(e) = " + $(e))
     log(DEBUG,"DEBUG: e  = " + e)

     $number = extract((e), "^instances\.([0-9]+)\.Id")
     log(DEBUG,"DEBUG: Found token " + $number)

     if ( int($number) >= 0 )
     {
        # Set token
        instances_Id[$number] = $(e)
        log(DEBUG,"DEBUG: instances_Id" +  "[" + $number + "]=" + instances_Id[$number])
     }
   }


# instances.#.Name
   if(regmatch((e),"^instances\.[0-9]+\.Name"))
   {
     log(DEBUG,"DEBUG: $(e) = " + $(e))
     log(DEBUG,"DEBUG: e  = " + e)

     $number = extract((e), "^instances\.([0-9]+)\.Name")
     log(DEBUG,"DEBUG: Found token " + $number)
```

```
     if ( int($number) >= 0 )
     {
        # Set token
        instances_Name[$number] = $(e)
        log(DEBUG,"DEBUG: instances_Name" +  "[" + $number + "]=" +
instances_Name[$number])
     }
   }

# instances.#.Location
   if(regmatch((e),"^instances\.[0-9]+\.Location"))
   {
     log(DEBUG,"DEBUG: $(e) = " + $(e))
     log(DEBUG,"DEBUG: e  = " + e)

     $number = extract((e), "^instances\.([0-9]+)\.Location")
     log(DEBUG,"DEBUG: Found token " + $number)

     if ( int($number) >= 0 )
     {
        # Set token
        instances_Location[$number] = $(e)
        log(DEBUG,"DEBUG: instances_Location" +  "[" + $number + "]=" +
instances_Location[$number])
     }
   }


  }
}

log(INFO,"CHECK: max_instance = " + $max_instance)

# Debugging - work through arrays
#
# Use the foreach with genevent to create events
# as required per array row
# e.g.
# @AlertGroup = "MessageBus"
# @AlertKey   = instances_Id[$count]
# @Node       = instances_Name[$count]
# @Location   = instances_Location[$count]
# @Severity   = 2
# @Identifer  = @Node + ":" + @AlertKey + ":" + @AlertGroup + ":" + @Type + ":" + @Agent
+ ":" + @Manager
# genevent(DefaultAlerts)
#
# Where
# DefaultAlerts = registertarget( %Server, %ServerBackup, "alerts.status" )


  $count=0
  foreach (i in instances_Id)
  {
    log(DEBUG,"DEBUG: instances_Id[" + $count + "] = " + instances_Id[$count])
    log(DEBUG,"DEBUG: instances_Name[" + $count + "] = " + instances_Name[$count])
    log(DEBUG,"DEBUG: instances_Location[" + $count + "] = " +
instances_Location[$count])
    $count = int($count) + 1
  }
# EOF
```

## 4.12 WebSocket connecting to STOMP

The STOMP services support WebSocket connections, and forwards JSON with a text header.

For the WebSocket transport the probe constructs the URI as.

```
WS://Host:Port/webSocketURI
```

The secure WebSocket is enabled when the probe property settings are used.

```
# SSL
EnableSSL            : 'true'
KeyStore             : '$NCHOME/omnibus/probes/certs/keystore.jks'
KeyStorePassword     : 'netcool'
```

The URI becomes.

```
WSS://Host:Port/webSocketURI
```

### 4.12.1 Property file

Property file needs Username and password for basic authentication.
The Host and Port property settings are used in the construction of the WebSocket URI.

```
# STOMP server settings
Host                    : '<STOMP-SERVER-IP>'
Port                    : <STOMP-PORT>
Username                : '<stomp-user>'
Password                : '<stomp-password>'
# Transport
TransportType           : 'WebSocket'
TransportFile           :
'$NCHOME/omnibus/probes/linux2x86/webSocketTransportSTOMP.properties'
# JSON parsing
MessagePayload          : 'json'
JsonMessageDepth        : 10
# STOMP Messages JSON Parser
TransformerFile         :
"$NCHOME/omnibus/probes/linux2x86/message_bus_parser_stomp.json"
# Best practice
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
```

### 4.12.2 Transport properties

The Message bus probe only allows the probe to SUBSCRIBE with the WebSocket protocol.

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

```
# Use basic authentication to connect
httpHeaders=Authorization=Basic ++Username++:++Password++

# STOMP messages endpoint
webSocketURI=/stomp

# STOMP CONNECT and SUBSCRIBE for messages
webSocketSubscribeMessage=CONNECT\naccept-version:1.2,1.1\nheart-
beat:10000,10000\n\n^@SUBSCRIBE\nd:sub-999\n
destination:/user/topic/updates\nack:auto\n\n^@SEND\naccept-version:1.2\n\n^@

# Ping WebSocket
webSocketRefreshMessage=Ping
webSocketRefreshInterval=10

# EOF
```

### 4.12.3 Transformer file

The transformer JSON parser file is used to strip away the STOMP header text, which is not parsed by the message bus probe, as it only accepts JSON or XML text data.

File : stomp_parser_config.json

```
{
  "eventSources" : [ {
    "name" : "OtherAlarmParser",
    "type" : "ANY",
    "config" : {
      "dataToRecord" : [ ],
      "messagePayload" : "json",
      "messageHeader" : "",
      "jsonNestedPayload" : "",
      "jsonNestedHeader" : "",
      "messageDepth" : 10,
      "headerPrefix" : "",
      "preparserScript" :
"/opt/IBM/tivoli/netcool/omnibus/probes/linux2x86/stomp_preparser.js"
    }
  }]
}
```

### 4.12.4 Java Script Pre-Parser

The file defined by the preparserScript allows the incoming data to be converted to JSON data using Java Script.

File : $NCHOME/omnibus/probes/linux2x86/stomp_preparser.js

```
/*
 * Java script to remove non-JSON
 */

/*
 * WatsonX function to remove unwanted lines
 */
function removeNonJsonLines(message) {
  const lines = message.split('\n');
  const jsonLines = lines.filter(line => {
    try {
      JSON.parse(line);
      return true;
    } catch (e) {
      return false;
    }
  });
  return jsonLines.join('\n');
}
/*
 * Remove unwanted lines from JSON message
 */

function preParser(input, endpoint) {

input = removeNonJsonLines(input);

return(input);

}
```

# 5 WebhookV2 transport

The WebhookV2 Transport provides a multichannel solution for the Webhook transport.

The main configuration file is the restMultiChannelHttpTransport.json, specified by the httpConnectionPropertiesFile setting, which is copied to a suitable location, and edited with the custom settings.

The main settings in the configuration are the first four parameters, in the action.

```
"uri":"http|https://HOST:PORT/URL_PATH",
"method":"POST|GET",
"headers":"HEADER#1,HEADER#2,HEADER#3",
"content":"CONTENT",
```

These are set as the standard Webhook settings are set.
With the URI requiring explicit HTTP|HTTPS, HOST and PORT settings, which allows the probe to connect to more than one host and port. Useful for systems with distributed services.

The other settings are common as well, with the 'interval' being set to 0, when not required.

```
"contentFile" : "",
"interval":"0",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

For refresh the 'interval' settings, prime numbers should be used, with the chosen values being unique for the probe instance.

The 'requireSSL' and 'enableHostnameVerification' are useful for turning off SSL security.
For example, if SSL is not required HTTP is used in the 'uri' and 'requireSSL' is set to 'false'.
If the SSL certificates do not include suitable Subject Alternate Names, the 'enableHostnameVerification' can be set to 'false', to turn-off the SSL hostname checking.

## 5.1 GLOBAL

The global settings are used to configure the general settings used by the HTTP requests. These settings include the keepTokens, autoReconnect and securityProtocol settings. Enabling 'autoReconnect' is useful when the server is prone to disconnecting clients. The 'httpHeaders' is used for general HTTP header requirements, for all action types. Only 'httpVersion' 1.1 is supported presently.

```
"httpVersion":"1.1",
"httpHeaders":"",
"responseTimeout":"60",
"securityProtocol":"TLSv1.2",
"keepTokens":"token, access_token, refresh_token",
"autoReconnect":"ON",
"gatherSubsTopicInfo":"false"
```

## 5.2 LOGIN

The login section performs the server login and login token refresh requests. The settings are configured for SSL/TLS by default.

### 5.2.1 GET_ACCESS_TOKEN

```
"uri":"https://HOST:PORT/rest-gateway/rest/api/v1/auth/token",
"method":"POST",
"headers":"Authorization=Basic ++Username++:++Password+
+,Accept=application/json,Content-Type=application/json,Use-Cookie=true,User-
Agent=IBM Netcool/OMNIBus Message Bus Probe",
"content":"{\"grant_type\":\"client_credentials\"}",
"contentFile" : "",
"interval":"0",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

### 5.2.2 GET_REFRESH_ACCESS_TOKEN

```
"uri":"https://HOST:PORT/rest-gateway/rest/api/v1/auth/token",
"method":"POST",
"headers":"Authorization=Basic ++Username++:++Password++,
Accept=application/json,Content-Type=application/json,Use-Cookie=true,User-Agent=IBM
Netcool/OMNIBus Message Bus Probe",
"content":"{\"grant_type\":\"refresh_token\",\"refresh_token\":\"++refresh_token+
+\"}",
"contentFile" : "",
"interval":"60",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

## 5.3 OAUTH

There is a separate section of OAuth authentication. Even though OAuth authentication can be configured using the LOGIN section.

```
"uri":"",
"basicAuthenticationUsername":"",
"basicAuthenticationPassword":"",
"scopes":"read,write",
"clientId":"",
"clientSecret":""
```

## 5.4 OIDC

### 5.4.1 INTROSPECTION

```
"uri":"https://HOST:PORT/c2id/token/introspect",
"method":"POST",
"headers":"Authorization=Bearer ++access_token++,Accept=application/json,Content-
Type=application/x-www-form-urlencoded",
"content":"",
"contentFile" : "",
"interval":"600",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

## 5.5 SUBSCRIBE

### 5.5.1 GET_SUBSCRIPTION

```
"uri":"https://HOST:PORT/nbi-notification/api/v1/notifications/subscriptions",
"method":"POST",
"headers":"Authorization=Bearer ++access_token++,Accept=application/json,Content-
Type=application/json,Use-Cookie=true,User-Agent=IBM Netcool/OMNIBus Message Bus
Probe",
"content":"{\"categories\":[{\"name\":\"NSP-
FAULT\",\"propertyFilter\":\"severity='major'\"}]}",
"contentFile" : "",
"interval":"0",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

### 5.5.2 GET_SUBSCRIPTION_REFRESH

```
"uri":"https://HOST:PORT/nbi-notification/api/v1/notifications/subscriptions/+
+subscriptionId++/renewals",
"method":"POST",
"headers":"Authorization=Bearer ++access_token++,Accept=application/json,Content-
Type=application/x-www-form-urlencoded,Use-Cookie=true,User-Agent=IBM Netcool/OMNIBus
Message Bus Probe",
"content":"",
"contentFile" : "",
"interval":"120",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

## 5.6 RESYNC

For resynchronisation to occur the probe property 'InitialResync' needs to be set to 'true' or else the 'ResyncInterval' needs to be set to a value, in seconds.

### 5.6.1 RESYNC_CHILD_BY_FDN

```
"uri":"https://HOST:PORT/nfm-p/rest/api/v1/managedobjects/children",
"method":"POST",
"headers":"Authorization=Bearer ++access_token++,Accept=application/json,Content-
Type=application/json,Use-Cookie=true,User-Agent=IBM Netcool/OMNIBus Message Bus
Probe",
"content":"{\"fdn\":\"Access Ingress:1\",\"fullClassNameList\":[\"aingr.Queue\"]}",
"contentFile" : "",
"interval":"0",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

## 5.7 LOGOUT

### 5.7.1 REVOKE_ACCESS_TOKEN

```
"uri":"https://HOST:PORT/rest-gateway/rest/api/v1/auth/revocation",
"method":"POST",
"headers":"Authorization=Basic ++Username++:++Password++,
Accept=application/json,Content-Type=application/x-www-form-urlencoded,Use-
Cookie=true,User-Agent=IBM Netcool/OMNIBus Message Bus Probe",
"content":"token=++refresh_token++&token_type_hint=token",
"contentFile" : "",
"interval":"0",
"attempts":"0",
"requireSSL":"true",
"asEventStream" : "false",
"enableHostnameVerification" : "true"
```

# 6 The Probe Environment file

The message bus probe environment file or the users environment can be used to set specific options for the probe instances.

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

## 6.1 Custom Java

The NCO_PROBE_JRE environment variable is used to define a non-standard Java JRE.

```
###
# Custom Java
NCO_PROBE_JRE=/opt/jdk1.8.0_381/jre
echo "NCO_PROBE_JRE=$NCO_PROBE_JRE"
```

## 6.2 Proxy settings

The probes NCO_JPROBE_JAVA_FLAGS environment variable can be used to set the Java options for HTTP and HTTPS proxy settings.

Example settings.

```
###
# Enable PROXY server Host and Port for HTTPS and SOCKS
NCO_JPROBE_JAVA_FLAGS="-Dhttps.proxyHost=192.168.2.1 -Dhttps.proxyPort=8443 ${NCO_JPROBE_JAVA_FLAGS}"
NCO_JPROBE_JAVA_FLAGS="-Djdk.http.auth.tunneling.disabledSchemes='' ${NCO_JPROBE_JAVA_FLAGS}"
NCO_JPROBE_JAVA_FLAGS="-DsocksProxyHost=192.168.2.1 -DsocksProxyPort= 8443 ${NCO_JPROBE_JAVA_FLAGS}"
```

Full set of options.

http.proxyUser
http.proxyPassword
http.proxyHost
http.proxyPort

https.proxyUser
https.proxyPassword
https.proxyHost
https.proxyPort

socksProxyHost
socksProxyPort
java.net.socks.username
java.net.socks.password

## 6.3 Enforcing Ipv4 or IPv6

The Webhook transport can be forced to use Ipv4 or Ipv6 using the java.net options.

```
###
# Custom settings
###
# Need IPv6 to be used by probe
#NCO_JPROBE_JAVA_FLAGS="-Djava.net.preferIPv6Addresses=true $NCO_JPROBE_JAVA_FLAGS"
# OR
###
# Need IPv4 to be used by probe
NCO_JPROBE_JAVA_FLAGS="-Djava.net.preferIPv4Stack=true $NCO_JPROBE_JAVA_FLAGS"
###
# Debug checking
echo "NCO_JPROBE_JAVA_FLAGS=$NCO_JPROBE_JAVA_FLAGS"
# EOF
```

# 7 Evaluating a REST service

In order to evaluate the whether the Message bus probe can connect to a service check the following features.

## 7.1 Host and Port

The probe server needs to be able to connect to the service using the required host and port.
Curl can be used to perform this check.

Use the hostname or IP address required by the REST service.

```
sh
export HOST PORT
HOST=localhost
PORT=8080
curl -v telnet://${HOST}:${PORT}
```

Successful results.

```
*   Trying ::1:8080...
Connected to localhost (::1) port 8080 (#0)
```

Using openssl to check connectivity,

```
openssl s_client -connect ${HOST}:${PORT}
```

Successful results.

```
Connecting to ::1
CONNECTED(00000003)
```

## 7.2 SSL details

Use openssl to check the connectivity details to a HTTPS/SSL port.
The certificates and ciphers are provided with the openssl output.

```
sh
export HOST PORT
HOST=localhost
PORT=8443
openssl s_client -connect ${HOST}:${PORT}

Successful results.

Connecting to ::1
CONNECTED(00000003)
.
<certificate details>
Server certificate
-----BEGIN CERTIFICATE-----
.
-----END CERTIFICATE-----
.
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: ...
```

The certificates can be placed into a file, and their details checked using keytool.

e.g.
File : server.pem

```
-----BEGIN CERTIFICATE-----
.
-----END CERTIFICATE-----
```

Command:

```
keytool -printcert -file server.pem
```

Results:

```
Owner: CN=webhook, OU=Netcool, O=IBM, L=New York, ST=New York, C=US
Issuer: CN=rootca, OU=Netcool, O=IBM, L=New York, ST=New York, C=US
Serial number: ...
Valid from: Mon Nov 04 07:12:09 EST 2024 until: Thu Nov 02 08:12:09 EDT 2034
.
SubjectAlternativeName [
  DNSName: localhost
  IPAddress: 0.0.0.0
  IPAddress: 127.0.0.1
]
```

To check the certificates use.

```
openssl s_client -host $HOST -port $PORT -showcerts
```

Successful results.
Same as the connect including the related certificates.

## 7.3 Curl check

Once the connectivity is confirmed use the curl command to examine the commands and the servers response.
Ensure that the server connection occurs and supports the available versions of the probe.

TLS version is defined by the version of Java being used by the message bus probe.
The message bus probe only supports HTTP 1.1.

Use the curl command with the required options, to confirm connectivity.

**Test command:**

```
curl -v \
--http1.1 \
--tlsv1.2 \
-k \
-u USERNAME:PASSWORD \
-X POST 'https://localhost:8443/restconf/auth' -d '{"Message":"Test"}'
```

Example results:

```
*   Trying ::1:8443...
* Connected to localhost (::1) port 8443 (#0)
* ALPN, offering http/1.1
*  CAfile: /etc/pki/tls/certs/ca-bundle.crt
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Unknown (23):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Unknown (23):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
...
* Server auth using Basic with user 'USERNAME'
* TLSv1.2 (OUT), TLS header, Unknown (23):
> POST /restconf/auth HTTP/1.1
> Host: localhost:8443
> Authorization: Basic VVNFUk5BTUU6UEFTU1dPUkQ=
> User-Agent: curl/7.x.x
> Accept: */*
> Content-Length: 18
> Content-Type: application/x-www-form-urlencoded
>
* TLSv1.2 (IN), TLS header, Unknown (23):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.2 (IN), TLS header, Unknown (23):
...
< HTTP/1.1 204 No Content
```

**Conclusion:**
The server supports HTTP 1.1 and TLSv1.3 with cipher TLS_AES_256_GCM_SHA384.

# 8 Troubleshooting

## 8.1 Checking CURL commands

The netcat command can be used to inspect data.

Convert the CURL commands to be sent to a localhost port that has netcat listening on, and send the CURL command there, to see what the data looks like.

```
curl --location -v \
--request POST 'http://localhost:8443/restconf/auth' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'user=netcool' \
--data-urlencode 'password=users_password'
```

Start netcat on another terminal.

```
nc -l 8443
```

Then send the CURL command.

```
POST /restconf/auth HTTP/1.1
Host: localhost:8443
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

user=netcool&password=users_password
```

Here you can see the login details need to be.

```
loginRequestURI=/restconf/auth
loginRequestMethod=POST
loginRequestContent=user=netcool&password=users_password
loginRequestHeaders=
```

## 8.2 Debugging SSL Connectivity

The Java SSL debug logging can be enabled using the `NCO_JPROBE_JAVA_FLAGS and the probes env file.`

```
cd $NCHOME/omnibus/probes/java
vi nco_p_message_bus.env
# SSL Handshake logging
NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=ssl:handshake:verbose $NCO_JPROBE_JAVA_FLAGS"
# Full handshake logging
# NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=all:handshake:verbose $NCO_JPROBE_JAVA_FLAGS"
echo "NCO_JPROBE_JAVA_FLAGS=$NCO_JPROBE_JAVA_FLAGS"
# EOF
```

The SSL logging is sent to the command line but this can be redirected to a file as shown.

```
cd $NCHOME/omnibus/prbes/linux2x86
../nco_p_message_bus -propsfile ./message_bus.props >
$NCHOME/omnibus/log/message_bus.ssl.log
```

Checking the log for found certificates:

```
grep -i found /tmp/message_bus.ssl.log
```

Checking the log for key checkpoints in the SSL communication:

```
grep '\*\*\*' /tmp/message_bus.ssl.log
```

Checking the log for Common Names:

```
grep 'CN=' /tmp/message_bus.ssl.log
```

Checking the probes certificate

```
openssl s_client -connect host:port
```

## 8.3 Creating ProbeKeyStore.jks

The probe keystore can be created using openssl on Linux using the following script.

```
File : createProbeStore.sh

#! /bin/sh
#####################################################
# Description
# Create SSL certificates and JKS store file for
# a client [probe] based on the FQDN
#####################################################
# Command JAVA_HOME
JAVA_HOME=$NCHOME/platform/linux2x86/jre64_1.8.0/jre
#####################################################
# Probe Server FQDN
#####################################################
PROBEFQDN="<FQDN>"
#####################################################
# SSL parameters
#####################################################
# Set a common passphrase for everything for now
PASSPHRASE="netcool"
MYOU="Netcool"
MYO="IBM"
MYCITY="New York"
MYSTATE="New York"
MYCC="US"
#####################################################
export MYOU MYO MYCITY MYSTATE MYCC PASSPHRASE
export PROBEFQDN
#####################################################
# Set path to common java
#####################################################
PATH=JAVA_HOME/jre/bin:$PATH
#####################################################
# Clean up old files
#####################################################
echo "*** Cleaning up old files"
rm rand cakey.pem careq.pem caroot.cer
rm ProbeKeyStore.jks
#####################################################
# Create ROOT CA
#####################################################
echo "*** Create CA Certificate"
RANDFILE=rand
export RANDFILE
openssl req -new -keyout cakey.pem -out careq.pem <<EOF
${MYCC}
${MYSTATE}
${MYCITY}
${MYO}
${MYOU}
${PROBEFQDN}
root@${PROBEFQDN}
${PASSPHRASE}
${MYO}
EOF
```

```
openssl x509 -signkey cakey.pem -req -days 3650 -in careq.pem -out caroot.cer -extensions
v3_ca

ls -l

echo "*** Creating ProbeKeyStore"

keytool -keystore ProbeKeyStore.jks -alias probe -validity 3650 -genkey -keyalg RSA \
-ext SAN=DNS:${PROBEFQDN} -storepass ${PASSPHRASE} <<EOF
${PROBEFQDN}
${MYOU}
${MYO}
${MYCITY}
${MYSTATE}
S{MYCC}
yes
${PASSPHRASE}
EOF

keytool -keystore ProbeKeyStore.jks -alias probe -certreq -file probehost.unsigned.cert
-storepass ${PASSPHRASE}

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

keytool -keystore ProbeKeyStore.jks -alias ROOTCA -import -file caroot.cer -storepass $
{PASSPHRASE}
keytool -keystore ProbeKeyStore.jks -alias probe -import -file probehost.signed.cert
-storepass ${PASSPHRASE}

echo "*** List results"

keytool -printcert -v -file caroot.cer
keytool -list -keystore ProbeKeyStore.jks -storepass ${PASSPHRASE}

echo "*** Created files"
ls -l

#EOF
```

### 8.3.1 Setting the probes truststore

The default truststore is the cacerts used by Java, and is not normally required to be set.
e.g.

```
find $NCHOME -name  cacerts
$NCHOME/platform/linux2x86/jre64_1.8.0/jre/lib/security/cacerts
$NCHOME/platform/linux2x86/jre_1.8.0/jre/lib/security/cacerts
```

If the Java Home is set explicitly then it's cacerts will be used.
Check the probes debug log file to confirm the truststore being used.

The truststore can be overridden using the Java options set in the probes env file.
e.g.

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

#|Java truststore
NCO_JPROBE_JAVA_FLAGS=
"-Djavax.net.ssl.trustStore=$NCHOME/omnibus/probes/truststores/ProbeKeyStore.jks
-Djavax.net.ssl.trustStorePassword=password $NCO_JPROBE_JAVA_FLAGS"
```

## 8.4 Transport debug log

```
Directory : $NCHOME/omnibus/java/conf/log4j
File : log4j2.xml

<?xml version="1.0" encoding="UTF-8"?>
<!-- Log4j2 Configuration for Cometd Transport -->
<Configuration status="warn">
  <Appenders>
    <File name="LOGFILE" fileName="${env:OMNIHOME}/log/transport.log">
      <PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="debug" >
      <!-- Increase the logging level if necessary  -->
      <AppenderRef ref="LOGFILE"/>
    </Root>
  </Loggers>
</Configuration>
```

You can use the transport.log file to review the transport specific details.

## 8.5 Unique Transport debug log

This method allows the probe to create unique names for all the extra logging.
The log4j2.xml can use the UNIQUENAME too.

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

```
# Debugging
# Create a unique log file name
export PROBENAME UNIQUENAME
PROBENAME=`echo $PROGRAM | awk -Fnco_p_ '{print $2}'`
UNIQUENAME=${PROBENAME}.$$
echo "UNIQUENAME=${UNIQUENAME}"

# Set debugging variables for SSL
# NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=ssl:handshake:verbose $NCO_JPROBE_JAVA_FLAGS"
# FOR ALL
NCO_JPROBE_JAVA_FLAGS="-Djavax.net.debug=all:handshake:verbose $NCO_JPROBE_JAVA_FLAGS"
# Non-native logging
NDE_DEFAULT_LOG_LEVEL="debug"
NDE_FORCE_LOG_MODULE="$NCHOME/omnibus/log/${UNIQUENAME}_forced.log"
NCO_P_NONNATIVE_TRANSCRIPT="$NCHOME/omnibus/log/${UNIQUENAME}_nonnative.log"
export NDE_DEFAULT_LOG_LEVEL NDE_FORCE_LOG_MODULE NCO_P_NONNATIVE_TRANSCRIPT

# Debug messages
echo "NCO_JPROBE_JAVA_FLAGS=$NCO_JPROBE_JAVA_FLAGS"
# EOF
```

File : $NCHOME/omnibus/java/conf/log4j/log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Log4j2 Configuration for Cometd Transport -->
<Configuration status="warn">
  <Appenders>
    <File name="LOGFILE" fileName="${env:OMNIHOME}/log/transport${env:UNIQUENAME}.log">
      <PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="debug" >
      <!-- Increase the logging level if necessary  -->
      <AppenderRef ref="LOGFILE"/>
    </Root>
  </Loggers>
</Configuration>
```

## 8.6 Reviewing the probes log

The Message bus probes debug log will use the standard log messages.
e.g.

```
grep Error: message_bus.log
grep Warning: message_bus.log
grep Information: message_bus.log
```

Along with these messages you can check for:

```
grep -i exception message_bus.log
grep -i latch message_bus.log
grep -i probewatch message_bus.log
```

Checking for connection types:

```
grep -i http message_bus.log
grep -i https message_bus.log
grep -i ws message_bus.log
```

## 8.7 Parsing events

The following examples are given to supplement the examples provided In the product manual.

### 8.7.1 JSON Nested events

Incoming events:

```
{\"status\":\"ok\",
\"Events\":[
{\"service\":\"node001\",\"state\":\"OK\"},
{\"service\":\"node002\",\"state\":\"OK\"},
{\"service\":\"node003\",\"state\":\"OK\"}
]}
```

Probe property settings:

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

Tokens:

```
Event#1:
service:    node001
state:      OK

Event#2:
service:    node002
state:      OK

Event#3:
service:    node003
state:      OK
```

with:

```
MessagePayload  : 'json'
```

Tokens:

```
Event#1:
status:     ok
Events.0.service:  node001
Events.1.service:  node002
Events.2.service:  node003
Events.0.state:    OK
Events.1.state:    OK
Events.2.state:    OK
```

## 8.8 Useful FAQs

There are useful FAQs that include examples and scripts for easier message bus probe configuration.

Using the message bus probe configuration as a HTTPS listener
https://www.ibm.com/support/pages/node/6563241

Message bus probe frequently asked questions
https://www.ibm.com/support/pages/node/7235817

Message bus probe with Kafka integrations frequently asked questions
https://www.ibm.com/support/pages/message-bus-probe-kafka-integrations-frequently-asked-questions

Message bus probe Webhook and WebSocket integrations frequently asked questions
https://www.ibm.com/support/pages/node/7179508

Twenty frequently asked questions for the message bus probe
https://www.ibm.com/support/pages/twenty-frequently-asked-questions-message-bus-probe

Netcool/OMNIbus 8.1 SSL frequently asked questions
https://www.ibm.com/support/pages/node/7150165

# 9 Known Issues

A number of known issues are fixed in the latest message bus probe.

## *9.1 DT465640 - Message bus probe - Webhook transport - Webhook accepts methods other than POST*

New message bus probe transport property allowPOSTOnly added for Webhook and WebhookV2 transports to restrict the Webhook listener to HTTP POST and HEAD only.

allowPOSTOnly=OFF|ON

allowPOSTOnly=OFF (default):

HTTP POST: Accepted
HTTP GET: Accepted
HTTP HEAD: Accepted
All other methods (OPTIONS, DELETE, TRACE, etc.): Return HTTP 405 Method Not Allowed

```
allowPOSTOnly=ON
```

HTTP POST: Accepted
HTTP HEAD: Accepted
HTTP GET: Return HTTP 405 Method Not Allowed
All other methods: Return HTTP 405 Method Not Allowed

## *9.2 DT400518 -* Message bus probe Webhook event size cannot be controlled via the transport properties

The Webhook and WebhookV2 transports accept the setting of the maxContentLength.
The maximum content length from connecting HTTP clients can be configured from the transporties file.
The allowed range is from 18 bytes to 2147483647 bytes (2GB).

For example.

```
# Optimum message size
# maxContentLength=524288
# Old Default
# maxContentLength=10485760
# Old Maximum
# maxContentLength=20971520
# New Minimum
# maxContentLength=18
# New Maximum
# maxContentLength=2147483647
###
# Set to 2Mb
maxContentLength=2097152
```