meshIQ Streams Express Guide

Version 1.0.0

Document Title: meshIQ Streams Express Guide
Document Release Date: September 2023
Document Number: STEG 100.001

Published by:
Research & Development
meshIQ
88 Sunnyside Blvd, Suite 101
Plainview, NY 11803

ACKNOWLEDGEMENTS: THE FOLLOWING TERMS ARE TRADEMARKS OF MESHIQ IN THE UNITED STATES OR OTHER COUNTRIES OR BOTH: AUTOPILOT, AUTOPILOT M6, M6 WEB SERVER, M6 WEB CONSOLE, M6 FOR WMQ, MQCONTROL, NAVIGATOR, XRAY.

THE FOLLOWING TERMS ARE TRADEMARKS OF THE IBM CORPORATION IN THE UNITED STATES OR OTHER COUNTRIES OR BOTH: IBM, MQ, WEBSPHERE MQ, WIN-OS/2, AS/400, OS/2, DB2, INFORMIX, AIX, AND Z/OS.

JAVA, J2EE, AND THE JAVA LOGOS ARE TRADEMARKS OF SUN MICROSYSTEMS INC. IN THE UNITED STATES OR OTHER COUNTRIES, OR BOTH.

INSTALLANYWHERE IS A TRADEMARK OR REGISTERED TRADEMARK OF FLEXERA SOFTWARE, INC.

THIS PRODUCT INCLUDES SOFTWARE DEVELOPED BY THE APACHE SOFTWARE FOUNDATION (*HTTP://WWW.APACHE.ORG/*), INCLUDING DERBY DATABASE SERVER. THE JAKARTA PROJECT" AND "TOMCAT" AND THE ASSOCIATED LOGOS ARE REGISTERED TRADEMARKS OF THE APACHE SOFTWARE FOUNDATION.

INTEL, PENTIUM AND INTEL486 ARE TRADEMARKS OR REGISTERED TRADEMARKS OF INTEL CORPORATION IN THE UNITED STATES, OR OTHER COUNTRIES, OR BOTH.

MICROSOFT, WINDOWS, WINDOWS NT, WINDOWS XP, THE WINDOWS LOGOS, MICROSOFT SQL SERVER, AND MICROSOFT VISUAL SOURCESAFE ARE REGISTERED TRADEMARKS OF THE MICROSOFT CORPORATION.

UNIX IS A REGISTERED TRADEMARK IN THE UNITED STATES AND OTHER COUNTRIES LICENSED EXCLUSIVELY THROUGH X/OPEN COMPANY LIMITED.

MAC, MAC OS, AND MACINTOSH ARE TRADEMARKS OF APPLE COMPUTER, INC., REGISTERED IN THE U.S. AND OTHER COUNTRIES.

"LINUX" AND THE LINUX LOGOS ARE REGISTERED TRADEMARKS OF LINUS TORVALDS, THE ORIGINAL AUTHOR OF THE LINUX KERNEL. ALL OTHER TITLES, APPLICATIONS, PRODUCTS, AND SO FORTH ARE COPYRIGHTED AND/OR TRADEMARKED BY THEIR RESPECTIVE AUTHORS.

ORACLE, JAVA, AND MYSQL ARE REGISTERED TRADEMARKS OF ORACLE AND/OR ITS AFFILIATES.

OTHER COMPANY, PRODUCT, AND SERVICE NAMES MAY BE TRADEMARKS OR SERVICE MARKS OF OTHERS.

Contents

This page intentionally left blank.

Chapter 1: Introduction

Welcome to the *meshIQ Streams Express Guide*. This guide describes the installation and configuration steps required to collect data for analysis by meshIQ. Please review this guide carefully before using the software.

From application tracing and transaction tracking, meshIQ delivers powerful analysis of the data your business runs on, across the systems you rely on. This guide assumes that you have established a repository for the meshIQ Platform and that you have set up and configured it for the collection of the data. This book will guide you through the steps for collecting tracking data.

The purpose of meshIQ tracking is to analyze applications and determine their behavior. Here are some examples of its value:

1) **Tracking individual application calls**
    - Identify applications doing "unnecessary" calls as a result of inefficient logic.
    - Identify applications that are not conforming to messaging best practices.
    - Observe and compare the timings of calls based on different scenarios or environments.
2) **Analyze application behavior**
    - Identify Patterns.
    - Collect performance metrics.
    - Determine whether partition algorithms are effective
3) **Problem determination**
    - Evaluate what the application is actually doing.
    - Verify correct processing of error conditions and failures.

1.1 How This Guide is Organized

Chapter 1: General Introduction.

Chapter 2: Explains how to install meshIQ Streams Express.

Chapter 3: Discusses how to configure and start the data collector and how to configure which data will be collected.

History of This Document

Table 1-1. Document History

Release Date	Document Number	Version	Summary
July 2023	XR/DC 100.001	1.0	Initial draft.
September 2023	STEG 100.001	1.0	Terminology updated.

1.2 Intended Audience

This guide is intended for systems administrators and operating engineers responsible for the installation and configuration of the meshIQ environment.

1.3 Technical Support

For technical support, visit the *meshIQ Resource Center*.

Chapter 2: Initial Setup of meshIQ Streams Express

The meshIQ Streams Express zip file, **MeshIQStreamsExpress.zip**, contains everything you need to run the data collector. Copy it to the system you want to run the collector on.

After copying the file, unzip **MeshIQStreamsExpress.zip,** you will have *`install_dir/MeshIQStreamsExpress`*, containing samples and execution scripts.


NOTE

Please make sure that `JAVA_HOME` points to the installed version of Java.

The next step is to create a properties file that will define connectivity to your meshIQ repository. There are two ways to do this, as described in the section below.

2.1 Creating my_tnt4j.properties

In the run folder, the member **express_tnt4j.properties** is a template that can be used to create custom properties for your usage. The easiest way to do this is to run **express_setup.sh/cmd**. You can either supply the parameters or it will prompt you for them.

Syntax: express_setup.sh ***URL TOKEN***

URL: is the URL for the repository
Examples: *http://localhost:6580, https://12.34.5.6:8443, https://data.jkoolcloud.com*

TOKEN: is the access token related to the repository for storing data. This token must be a streaming token which allows you to put data into a repository.

Examples: *DefaultToken, da646ff4-8873-44e1-72f42a9ae764, MyTestToken*

On completion, you will have a new file called **my_tnt4j.properties** in the run folder.

Alternatively, you can manually copy **express_tnt4j.properties** to **my_tnt4j.properties**, making the changes as outlined.

2.1.1 TNT4J-Streams

For tracking, the functionality is based on tnt4j-streams. The tnt4j-streams folders contain a variety of samples that can be used with little or no modification to collect a variety of data and stream it to meshIQ tracking. Supports the following data sources:

- File
- Characters/bytes feed stream from file or over TCP/IP
- HDFS
- MQTT

  - HTTP
  - JMS
  - Apache Kafka (as Consumer and as Producer/Consumer interceptor)
  - Apache Flume
  - Logstash
  - WMQ (IBM MQ)
  - OS pipes
  - Zipped files (also applies for HDFS)
  - Standard Java InputStream/Reader
  - JAX-RS service (JSON/XML)
  - JAX-WS service
  - System command
  - MS Excel document
  - Elastic Beats
  - FileSystem (JSR-203 compliant) provided files (accessing remote files over SCP/SSH, SFTP, etc.)
  - JDBC
  - *Chronicle Queue*
  - Artemis Broker or Producer/Consumer interceptor
  - Compressed binary data or input stream
  - Protobuf messages

- Specialized parser use cases

  - Files (including provided by HDFS and JSR-203 FileSystem)
  - Apache Access Logs
  - IBM MQ Trace Events
  - IBM MQ Error log parser (also supports JSON formatted logs)
  - IBM Accounting and Statistics Data

2.1.2 Customizing a Sample Parser

The suggested method is to use a copy of the parser in the run folder. This allows for subsequent updates to tnt4j-streams and samples with minimal impact on your usage.

To do this, perform the following steps:

1. Look for an appropriate sample in `install_dir\MeshIQStreamsExpress\tnt4j-streams\samples`.

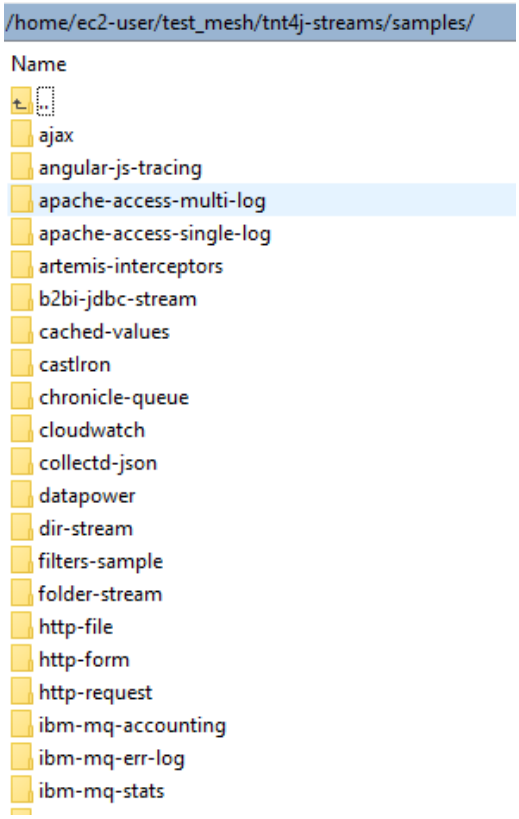


Figure A. \install_dir\MeshIQStreamsExpress\tnt4j-streams\samples

NOTE

For example, you would copy the \MeshIQStreamsExpress\tnt4j-streams\samples\ajax\ directory to \MeshIQStreamsExpress\run\ajax\, leaving the original files in their original locations.

2. Alter the files in the `install_dir\MeshIQStreamsExpress\`**`run`** subdirectory. Configure the parser as described in the comments within it as well as reference material in the README.md file. In most cases, the sample parser will be called tnt-data-source.xml. You can leave this name or change it to a more meaningful name.
3. Once the parser is configured, you can run it using the express_run.sh/cmd script.

   Syntax: express_run.sh ***FOLDER PARSER***

   FOLDER: is the folder name for the sample
   Examples: *ibm-mq-trace-events, single-log,elastic-beats*

   PARSER: is the parser to use. If not specified, the tnt-data-source.xml is used.

Several examples of parsers are included in the following sections.

Chapter 3: Sample Usage Examples

3.1 Tracking Log Files

There are many use cases for files; this section just covers a very basic one. Files (including those provided by HDFS and JSR-203 FileSystem) can be streamed.

- as "whole at once" - when a stream starts, it reads file contents line by line, so that a single file line holds the data for a single activity event. After file reading is completed, the stream stops.
- using file polling - when an application uses a file to write data at runtime, the stream waits for file changes. When there are file changes, the changed (appended) lines are read by the stream and interpreted as a single line is a single activity event. The stream stops only when the application is terminated or a critical runtime error occurs.

3.1.1 Copy and Review the Single File Sample

Copy the contents from **tnt4j-streams/samples/single-log** to the **run** folder (cp -rf tnt4j-streams/samples/single-log run). No changes are required, but you can review the member **run/single-log/tnt-data-source.xml**.

These are some of the key elements of this parser.

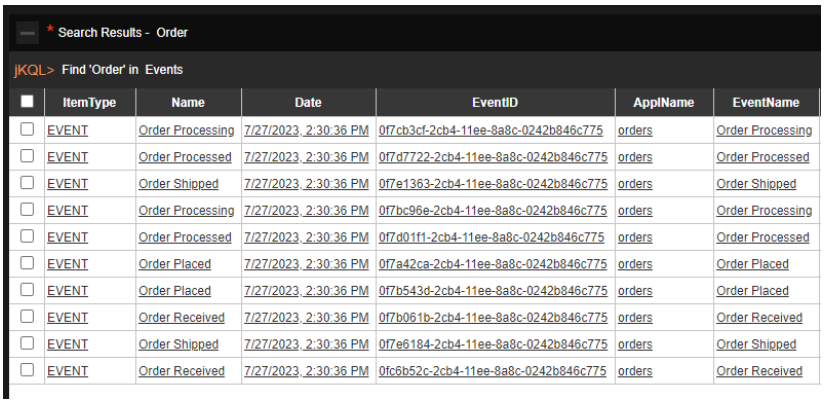
- The name of the log file to process
  ```
  <property name="FileName" value="orders.log"/>
  ```
- The delimiter that separates fields in the file
  ```
  <property name="FieldDelim" value="|"/>
  ```
- A field-by-field mapping of the contents of each field
  ```
  <field name="UserName" locator="4"/>
  ```
- A mapping rule to convert a simple value into more meaningful name
  ```
  <field name="EventType" locator="5">
              <field-map source="Order Placed"
  target="START"/>
  ```

3.1.2 Starting the collector for the Single File Sample

To start the data streaming process, from the run folder, execute either **express_run.sh single-log** or **express_run.cmd single-log**. Using the options supplied, it will connect to the file, parse the contents, and exit.

Some logging information, such as the fields parsed, is normal. Additional logging information can be found in the **single-logs/logs** folder. At this point, if your configuration is correct, you should receive 10 events related to orders processed in your repository.

You can find them easily by typing Order in the search field.



* Search Results - Order

jKQL> Find 'Order' in Events

	ItemType	Name	Date	EventID	ApplName	EventName
☐	EVENT	Order Processing	7/27/2023, 2:30:36 PM	0f7cb3cf-2cb4-11ee-8a8c-0242b846c775	orders	Order Processing
☐	EVENT	Order Processed	7/27/2023, 2:30:36 PM	0f7d7722-2cb4-11ee-8a8c-0242b846c775	orders	Order Processed
☐	EVENT	Order Shipped	7/27/2023, 2:30:36 PM	0f7e1363-2cb4-11ee-8a8c-0242b846c775	orders	Order Shipped
☐	EVENT	Order Processing	7/27/2023, 2:30:36 PM	0f7bc96e-2cb4-11ee-8a8c-0242b846c775	orders	Order Processing
☐	EVENT	Order Processed	7/27/2023, 2:30:36 PM	0f7d01f1-2cb4-11ee-8a8c-0242b846c775	orders	Order Processed
☐	EVENT	Order Placed	7/27/2023, 2:30:36 PM	0f7a42ca-2cb4-11ee-8a8c-0242b846c775	orders	Order Placed
☐	EVENT	Order Placed	7/27/2023, 2:30:36 PM	0f7b543d-2cb4-11ee-8a8c-0242b846c775	orders	Order Placed
☐	EVENT	Order Received	7/27/2023, 2:30:36 PM	0f7b061b-2cb4-11ee-8a8c-0242b846c775	orders	Order Received
☐	EVENT	Order Shipped	7/27/2023, 2:30:36 PM	0f7e6184-2cb4-11ee-8a8c-0242b846c775	orders	Order Shipped
☐	EVENT	Order Received	7/27/2023, 2:30:36 PM	0fc6b52c-2cb4-11ee-8a8c-0242b846c775	orders	Order Received

3.2 IBM MQ Activity Trace Events

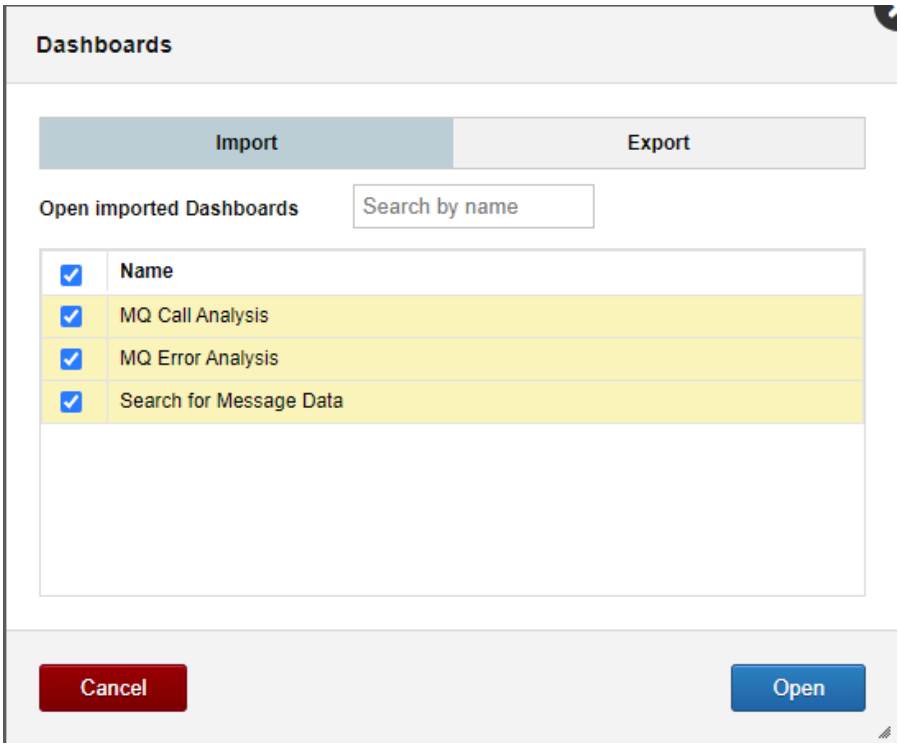
IBM MQ is a message system with a general mechanism for producing tracking events. meshIQ Streams Express can be installed on the MQ server or as an MQ client.

> NOTE
> The MQ client library is not required to be installed when remotely connecting to the queue manager; the package contains required client libraries.

There are two components to collecting the data: Application Activity Events must be generated, and a process must be configured to read the events.

3.2.1 Sample Dashboards

The folder **MeshIQStreamsExpress/dashboards** contains a set of sample dashboards, meshIQIbmmqDashboard.csv, using the dashboard import option. These are not required but speed up analysis of the data collected.



These can be used to do a quick analysis of the data collected:

- MQ Call Analysis – analyzes by type of call, queues used, and activity
- MQ Error Analysis – analyzes calls with non-zero return codes
- Search for Message Data – prompts for a string to find in message data (payload required)

3.2.2 IBM MQ Queue Manager Configuration

- Review the following section on how to collect MQ activity events. For further information, please see the following link:
- *https://www.ibm.com/docs/en/ibm-mq/9.3?topic=network-application-activity-trace*

3.2.2.1 Options for Activating IBM MQ Application Activity Events

There are three options for activating application activity events. Determine which option to use based on the specific case you want to track.

3.2.2.1.1 Activating at the Queue Manager Level

If you want to capture all activity within a queue manager, you can do this by setting the queue manager property ACTVTRC(ON). A common misunderstanding is that ACTVTRC(ON) is always required. If you know which applications to trace, it is better to include them in **mqat.ini** or use subscriptions and leave ACTVTRC(OFF).

While turning on at the queue manager is simple to do, it will collect a lot of information you will not want. This will include system activity, monitoring tools such as the MQ explorer, and other activity not tied to the applications to trace.

If you activate activity tracing at the queue manager level, it is important that you still use **mqat.ini** to exclude applications that you do not want to capture. To determine which applications to exclude, try the following:

1. Activate the trace for a period of time and then set it back off.
2. Review the MQ Calls by Application viewlet on the MQ Call Analysis Dashboard.
3. Identify application names that you do not want to collect.
4. Add these entries into the **mqat.ini** file for the queue manager(s) using a specific or generic reference. For example, nsq* would exclude any application beginning with nsq.

For this method, the messages are sent to the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. Follow the instructions in *Reading Trace Data from a Queue* to setup the parser.

3.2.2.1.2 Activating for Specific Applications Using mqat.ini

This method can be used when you have specific applications to track. The **mqat.ini** file controls which applications are included in tracing. As soon as you add an entry and save this member, the application will be traced the next time it connects to the queue manager. The **mqat.ini** file is found in the queue manager folder and can be edited with any text editor.


NOTE

This method cannot be used for the MQ appliance because only the default stanza can be changed, not application-specific entries. For information on selective tracing with the MQ Appliance, refer to *Activating Using a Subscription for Applications, Channels or Connections*.

There are two sections to this member. The default values are listed at the top of the file. The application-specific sections are at the bottom. As a general suggestion, set the defaults to the values you want to use for most applications.

The following is a recommended set of default values:

```
# Global settings stanza, default values
AllActivityTrace:
ActivityInterval=5
ActivityCount=100
TraceLevel=MEDIUM
TraceMessageData=1000
StopOnGetTraceMsg=ON
SubscriptionDelivery=BATCHED
```

This indicates:

- A single application trace entry will be written after 5 seconds or up to 100 MQ calls, whichever comes first. Higher numbers provide better performance but larger messages to process.
- The level of detail is MEDIUM, which satisfies most queries.
- Up to 1000 bytes of message payload will be captured. Capturing message payload is valuable to allow capturing specific application information. In some cases, you may not be permitted to capture payload.
- If the trace attempts to capture an application that is consuming trace events, the collection for that application is terminated (to eliminate infinite loops).
- Subscriptions are batched for delivery. This is relevant to the next section on subscriptions.

The second section of **mqat.ini** file contains the applications to include and exclude.

Examples:

```
ApplicationTrace:
  ApplName=amqsget*
  Trace=ON
ApplicationTrace:
  ApplName=amqsput*
  Trace=ON
```

The reason for using generics in these examples is that Windows application names include the .exe and to also include the client versions, **amqsgetc**.

You can also use generics to include a group of applications and then a more specific definition to exclude a portion of them.

For example:

```
ApplicationTrace:
  ApplName=amqs*
  Trace=ON
ApplicationTrace:
  ApplName=amqsact*
```

```
  Trace=OFF
```

This will trace all sample amqs applications but exclude amqsact.

You can also override a specific value here, such as changing the amount of payload data collected.

```
ApplicationTrace:
  ApplName=amqsput*
  Trace=ON
  TraceMessageData=500
```

When adding an entry for a running application, you may want to start tracing immediately rather than at the next startup. In this case, an "ALTER QMGR" command will trigger application tracing.

For this method, the messages are sent to the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE. Follow the instructions in *Reading Trace Data from a Queue* to setup the parser.

3.2.2.1.3 Activating Using a Subscription for Applications, Channels or Connections

This method is similar to using the **mqat.ini** file but instead of editing the **mqat.ini** file, you use subscriptions to specify which applications to trace. It additionally supports subscribing to a channel or a connection. This is the only method that can be used to trace selectively with the MQ appliance. All application activity for the MQ Appliance is client connection based. Being able to trace a channel means that there is no need to know which applications are running, which simplifies the configuration.

The subscriptions can be defined explicitly or dynamically.

- When creating explicit subscriptions, one of the benefits is that you control the destination(s), rather than using the SYSTEM queue. Follow the instructions in *Reading Trace Data from a Queue* to set up the parser specifying the name of your destination queue(s).
- Dynamic subscriptions are useful when doing ad hoc testing. To use dynamic subscriptions, Follow the instructions in *Using a Subscription to Gather Trace Data* to setup the parser. In this case, there are no queues to configure. However, no events will be recorded when the collector is not running since the subscription does not exist.

Another advantage of using subscriptions is that it takes effect immediately even for running applications, removing the need for altering the QMGR.

A disadvantage of this method is that traces are based on the default settings in **mqat.ini**. For example, all applications will collect the same amount of message content.

3.2.3 Modifying the Application Activity Events Parser

Copy the contents from **tnt4j-streams/samples/ibm-mq-trace-events** to the **run** folder:

```
cp -rf tnt4j-streams/samples/ibm-mq-trace-events run
```

Edit the member **tnt-data-source.xml**.

3.2.3.1 Reading Trace Data from a Queue

1. Locate the stream for queues, (approximately line 46)

```
<stream name="WmqActivityTraceStream" …
```

2. Change the following to match your queue manager name:

```
<property name="QueueManager" value="QMGR"/>
```

3. If using a queue other than the default SYSTEM queue, change this line to your queue name.

```
<property name="Queue"
value="SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE"/>
```

4. If connecting remotely to the queue manager and if credentials are required, supply the channel information, user, and password. If connecting locally to the queue manager, delete the channel, host, and port.

```
<!-- these are required if connecting remote or removed for
local connection -->
        <property name="Channel" value="[CHANNEL]"/>
        <property name="Host" value="[HOST]"/>
        <property name="Port" value="1414"/>

        <!-- user and password as required by the MQ Queue
Manager -->
        <!--<property name="UserName"
value="[USER_NAME]"/>-->
        <!--<property name="Password"
value="[USER_PASS]"/>-->
```

5. Review and supply additional options, such as those for SSL, as required.
6. Since you are not using a subscription, locate the lines starting <stream name="WmqActivityTraceStream2" and delete from there to the line </stream> (approximately lines 87-127).
7. (Optional.) A single instance of the meshIQ data collector can collect data from many queue managers, local or remote. If you want additional queue manager data, repeat the `<stream>` section as many times as required, setting a unique value for stream name for each (WmqActivityTraceStream3, etc.).. Alternately, you can connect to a single queue manager and forward events from all other queue managers to it using standard MQ routing techniques.
8. Save your changes.

3.2.3.2 Using a Subscription to Gather Trace Data

1. Locate the stream for subscriptions, (approximately line 88).

```
<stream name="WmqActivityTraceStream2" …
```

2. Change the following to match your queue manager name:

```
<property name="QueueManager" value="QMGR"/>
```

3. Change this topic string to match the subscription you need to create. Do not change the OpenOptions, which treats the * in the topic string as required.

```
<property name="TopicString"
value="$SYS/MQ/INFO/QMGR/[QMGR]/ActivityTrace/ApplName/amqs
*"/>
<property name="OpenOptions" value="MQSO_WILDCARD_CHAR"/>
```

4. If connecting remotely to the queue manager and if credentials are required, supply the channel information, user, and password. If connecting locally to the queue manager, delete the channel, host, and port.

```
<!-- these are required if connecting remote or removed for
local connection -->
        <property name="Channel" value="[CHANNEL]"/>
        <property name="Host" value="[HOST]"/>
        <property name="Port" value="1414"/>

        <!-- user and password as required by the MQ Queue
Manager -->
        <!--<property name="UserName"
value="[USER_NAME]"/>-->
        <!--<property name="Password"
value="[USER_PASS]"/>-->
```

5. Review and supply additional options, such as those for SSL, as required.
6. Since you are not using a queue, locate the lines starting <stream name="WmqActivityTraceStream" and delete from there to the line </stream> (approximately lines 45-85).
7. (Optional.) A single instance of the meshIQ data collector can collect data from many queue managers, local or remote. If you want additional queue manager subscriptions, repeat the WmqActivityTraceStream2 stream as many times as required, setting a unique value for stream name for each (WmqActivityTraceStream3, etc.).
8. Save your changes.

3.2.4 Starting the Collector for Application Activity Events

To start the data streaming process, from the run folder, execute either **express_run.sh ibm-mq-trace-events** or **express_run.cmd ibm-mq-trace-events**. Using the options supplied, it will connect to the queue manager(s) specified and wait for data. Some logging information such as queue manager connectivity is normal. If you see any error messages during startup, correct them and repeat. Additional logging information can be found in the ibm-mq-trace-events/logs folder.

Whenever you add additional queue managers to the parser or make any other changes, you will need to stop and restart the collector.

3.2.4.1 Verification

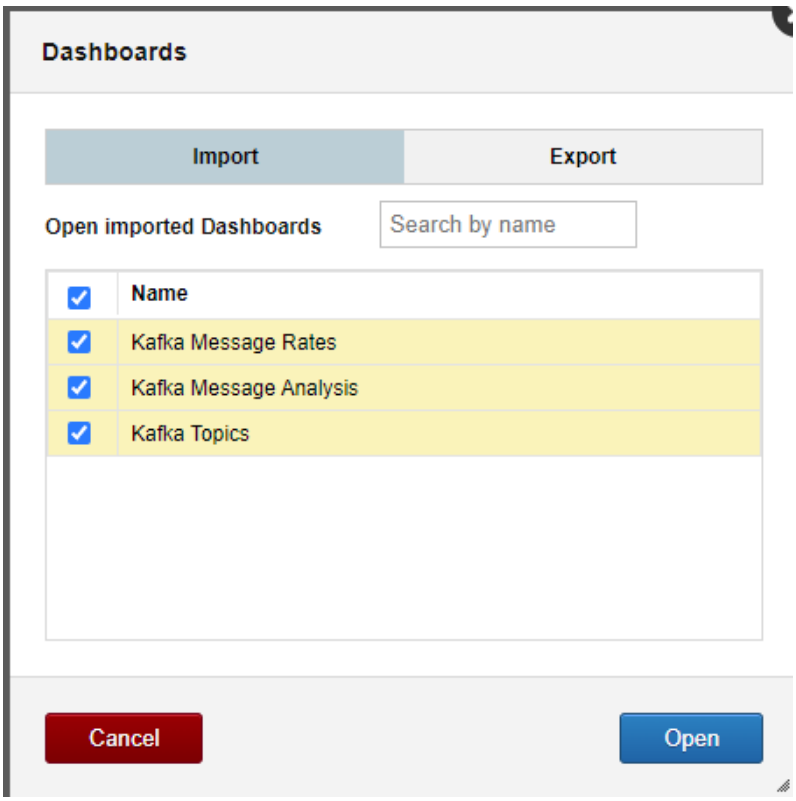
At this point, you should have data being collected in your repository. Allow a few minutes before returning to the GUI. Refresh the **MQ Call Analysis** dashboard within the GUI. It should now show data collected from your environment.

3.3 Kafka Topics

Apache Kafka is another common use case for tracking to handle data originating from Kafka. Unlike IBM MQ, for Kafka there is no setup required on Kafka server. The collector connects as a consumer to a topic and captures all messages flowing through it. The parser can elect to capture specific messages or content as required.

3.3.1 Sample Dashboards

The folder **MeshIQStreamsExpress/dashboards** contains a set of sample dashboard, **meshIQKafkaDashboard.csv**, for the tracking interface using the dashboard import option. These are not required but speed up analysis of the data collected.



These can be used to do a quick analysis of the data collected:

- Kafka Message Rates – analyzes the publish rates by topic
- Kafka Message Analysis – the messages and key sizes
- Kafka Topics – Analyzes the distribution of Kafka topics and partitions

3.3.2 Tracking a Single Kafka Topic

If you just want to track a few topics each with a unique parser, the following technique can be used. If you want to track a larger number of topics, the method described in *Tracking Multiple Kafka Topics* may be better.

3.3.3 Modifying the Single Kafka Topic Parser

Copy the contents from **tnt4j-streams/samples/kafka-client** to the **run** folder:

```
cp -rf tnt4j-streams/samples/kafka-client  run
```

Edit the member **tnt-data-source.xml**.

1. Specify the topic string that you want to listen to.
   <property name="Topic" value="tnt4j-streams-msg-topic"/>
2. If you have an existing configuration file for Kafka clients, you can specify it rather than re-entering information for steps 3 through 5.

   ```
   <property name="FileName" value=”my.properties"/>
   ```

3. Change the following as defined by your Apache Kafka setup.
   `<property name="bootstrap.servers" value="localhost:9092"/>`
4. Review the group.id. It must not conflict with any application consuming messages from this topic.
   `<property name="group.id" value="tnt4j-streams-kafka"/>`
5. Review and supply additional options, such as those for SSL, as required.
6. (Optional.) If you want additional topics, repeat the `<stream>` section as many times as required, setting a unique value for stream name for each. A single instance of the meshIQ data collector can collect data from many Kafka topics and brokers. Change the name of the client to be unique as well.
   `<property name="client.id" value="tnt4j-streams-kafka-consumer-stream"/>`
7. Save your changes.

3.3.4 Starting the Collector for Single Kafka Topics

To start the data streaming process, from the run folder, execute **express_run.sh(cmd) kafka-client**. Using the options supplied, it will connect to the broker(s) specified and wait for data.

Some logging information such as kafka connectivity is normal. If you see any error messages during startup, correct them and repeat. Additional logging information can be found in the **kafka-clients/logs** folder.

At this point, if your configuration is correct, you should start receiving data in your repository. Use the sample dashboards created above for verification.

Whenever you add additional Apache Kafka topics, brokers, or make any other changes, you will need to stop and restart the collector.

3.3.5 Tracking Multiple Kafka Topics

This parser is derived from the samples and shows how to simplify creating several parser instances. It takes a list of topics and automates the creation of a parser file to track them. It currently is available for Linux systems only.

You can listen to all topics if required, by using the topics --list request and using that list or a subset as input into the Kafka collector discussed in the section below (*Creating a List of Topics*).

3.3.6 Creating a List of Topics

Switch to the folder **run/kafka-topics** and edit the member **topics.txt**. This is a list of topics that you want to track. Change the sample names included to the list of topic names required and save the file. You can start with a list of all topics by doing a **topics --list** command and sending the output to a file and removing topics from it.

```
# place topics to listen to, one per line
my-topic
sample-topic
```

3.3.7 Starting the Collector for Multiple Topics

From the **run/kafka-topics** folder execute **run_topic_list.sh** to start capturing messages produced to these topics. The format of the command is:

Syntax run_topic_list.sh ***connection_info topic_list_file label***

connection_info is the broker connection string (default: localhost:9092)
topic_list_file is the file described above (default: topics.txt)
label is any identifier you want included with the data collected (default: same as connection_info)

Examples:

- run_topic_list.sh
- run_topic_list.sh kafka_server:9092
- run_topic_list.sh kafka_server:9093 mytopics.txt MyKafkaTopics

If you see any error messages during startup, correct them and repeat. Additional logging information can be found in the kafka-topics/logs folder.

If you need to add additional connection parameters such as SSL connectivity, you can add them in the file **models/express_kafka_stream.model**. See the example above for Single Kafka topics for additional parameters.

At this point, if your configuration is correct, you should start receiving data in your repository. Use the sample dashboards created above for verification.

Whenever you add additional Apache Kafka topics or make any other changes, you will need to stop and restart the collector.