Interaction Server Administration Guide

Interaction Management 8.1.4

1/4/2022

Table of Contents

Interaction Server Administration

This section provides information for administrators regarding Interaction Server

In addition to the information on this page, there is also information on:

- Limitations to observe concerning Interaction Server.
- Improving the performance of the Interaction Server database.
- Converting attached data to and from BLOB format.
- Deploying and using Event Logger, which stores reporting event messages in a database.

Be aware of the following:

- Use CC Pulse to monitor interaction queues (in interaction workflows) for signs of problems with routing strategies. If the number of interactions in a queue increases abnormally, it may be a sign that the strategy that processes interactions from that queue is not loaded in Universal Routing Server.
- Depending on the amount of configuration objects and the volume of the interactions stored in the Interaction Server database, it might take considerable time for Interaction Server to start up and shut down.
- Interaction Server has two possible Application types in the Configuration Layer. `Interaction Server` is the normal type; the `T-Server` type is also available for backward compatibility. Be aware that an Interaction Server 7.6 or later of type `T-Server`, upon startup, will make two attempts to connect to Configuration Server. The first attempt will generate `trace`-level alarms (about a missing application of type: Interaction Server) that you should ignore. The second attempt will succeed.
- If you want to use the Dynamic Workflow Management functionality, be sure to run Interaction Server with a user that has write access to the Configuration Server database for all of the tenants associated with this Interaction Server (that is, the user specified on the Security tab of the Interaction Server Application object).
  In this situation Interaction Server does not support Configuration Server Proxy, which has only read access to the Configuration Server database.

Interaction Server Limitations

- Interaction Server does not support the following requests:
  - RequestQueryServer
  - RequestQueryLocation
  - RequestDeletePair (when URS sends this request after RequestRouteCall)
- It is not desirable to run Interaction Server in an environment in which servers and clients differ as to the codepages used (by operating systems or databases). In such an environment, characters of non-Latin alphabets may appear as the symbol ? (question mark) in log files and in applications with a user interface, such as Agent Desktop. The functionality of other features of the solution may also be restricted or compromised.
- Making an on-the-fly change to the host or port specification (on the `Server Info` tab) of a backup Interaction Server will cause it to exit.
- When Interaction Server sends a database request right before disconnecting from DB Server, and the request executes after disconnecting, Interaction Server fails to generate events to clients for submitted interactions.
- Starting in release 8.1.3, the scripts supplied with Interaction Server for Oracle databases create the `flexible_properties` field with the type BLOB. To support this feature, you must use DB server 8.1.1 and above with Oracle client 10.2 and above.
- You cannot use commas (,) and semicolons (;) in interaction queue names.

Improving the Performance of the Interaction Server Database

To optimize the performance of Interaction Server, try the following steps:

1. Design or redesign the Business Process for greater efficiency; for example, by minimizing the number of processing steps. This provides for the most performance gain for custom Business Processes.
2. Analyze and optimize the SELECT statements generated by Interaction Server. Analyze the execution plans for the generated SELECT statements and create appropriate indexes. This is especially important if you have added an custom Business Processes: the standard indexes provided with the default schema do not take account of any custom database fields, specific conditions configured, and other items added by custom Business Processes. This step might provide all the performance gain that you need.
3. Perform a general tuneup on the database.
4. Partition the database. The subtopics in this section deal with this.

General Remarks on Partitioning

A partition is a division of a logical database or its constituent elements into independent parts. Database partitioning may be done for reasons of performance, manageability, or availability. This section concentrates on partitioning to improve performance.

By splitting a large table into several smaller tables, queries that need to access only a fraction of the data can run faster because there is less data to scan. Maintenance tasks, such as rebuilding indexes or backing up a table, can also run more quickly. Placing logical parts on physically separate hardware provides a major performance boost since all this hardware can perform operations in parallel.

Interaction Server performs large numbers of queries, updates, inserts, and deletes on its database. While it is relatively easy to achieve optimal performance with updates, inserts, and deletes, queries (SELECTs) are different.

The Interaction Server database consists of a single major table that stores all the interaction data. Every interaction in the system is always assigned to some interaction queue, represented by value of the field queue in the Interaction Server table. Business processes may employ dozens or even hundreds of queues.

Queues can vary greatly in the way they are used: some hold many interactions which are rarely processed at all (for example, an archive queue), others hold a small number of interactions with a high processing rate (for example, a queue for interactions that need some preliminary processing).

If these two types of queue are separated into different partitions, then the slower selection rate of the first type will not interfere with the high-speed selections of the second type. So the queue field is a natural choice to partition the data on. The remainder of this section describes partitioning by queue.

Planning

Decide for which queues it makes sense to separate data into logical partitions. Start by surveying the queues in your Business Processes and separate them out into three types:

1. Queues that contain high numbers of interactions; for example, post processing backlog or archive queues.
2. Queues that should not contain lots of interactions because all interactions in these queues should be processed immediately. A good example is the first queue in a Business Process which is meant for some preliminary processing (such as performing classification, calculating and attaching some user data, or sending an acknowledgment).
3. Queues that feed strategies that wait for resources (agents) to become available; usually there is a single such distribution queue in a Business Process.

Here is the rationale for separating data into at least three partitions that correspond to these three types of queues:

1. Separating Type 1 queues, those with many "inactive" interactions, ensures that these interactions are not even considered when SELECT statements are executed to pull interactions from Type 2 ("active") queues. Even if there are complex conditions for some views in your Business Process, there is much less data to scan because the majority of the interactions in an archive or post processing backlog are not touched by these scans.
2. Separating Type 2 queues is logical because most of the time these queues should be completely empty. Selecting new interactions out of these queues is trivial since there are not many interactions to select from.
3. Type 3 is the most demanding. While the rate of processing can be high, if there are many agents and handling time is relatively low, interactions may still accumulate in these queues when the peak inbound rate is higher than the processing rate. This means that SELECT statements are executed frequently against many records. If there are multiple queues of this type, it may be beneficial to assign them to separate partitions.

Hardware Planning

While purely logical separation of data may be of some benefit, placing the partitions on separate hard drives provides the best performance gain. In planning which drives in your system to dedicate to Interaction Server database partitions, it is advisable set aside one drive for the operating system and one for database log files, and place the Interaction Server database partition on other drives.

The rest of this section presents an example of partitioning using Microsoft SQL.

Creating the Database

To create a database with several file groups that will hold data for different partitions, use an SQL statement similar to the following:

```
CREATE DATABASE [itx_partitioned] ON PRIMARY
(NAME = N'itx802partitioned', FILENAME =
N'D:\MSSQL\DATA\itx802partitioned.ndf',
SIZE = 44828672KB, MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB),
FILEGROUP [P1]
(NAME = N'itx802partitioned1', FILENAME =
N'E:\MSSQL\DATA\itx802partitioned1.ndf',
SIZE = 2048KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB),
FILEGROUP [P2]
(NAME = N'itx802partitioned2', FILENAME =
N'F:\MSSQL\DATA\itx802partitioned2.ndf',
SIZE = 2048KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB),
FILEGROUP [P3]
(NAME = N'itx802partitioned3', FILENAME =
N'G:\MSSQL\DATA\itx802partitioned3.ndf',
SIZE = 2048KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
LOG ON
(NAME = N'itx802partitioned_log', FILENAME =
N'H:\MSSQL\DATA\itx802partitioned_log.ldf',
SIZE = 5095872KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
```

Or you can create the database and file groups using Microsoft SQL Management Studio.

You can create as many file groups as your resources allow.

Carrying Out the Partitioning

Partition Function

The partition function calculates the logical partition number for any specific record based on the record's field value. We only need to consider the value of the 'queue' field since we want to partition data according to queues.

The following is an example of the partition function:

```
CREATE PARTITION FUNCTION [QNamePFN](varchar(64)) AS
RANGE RIGHT FOR VALUES (N'Archive', N'Distribution', N'Inbound')
GO
```

Note that an SQL server partition function is always a range function. The values for the range function must be sorted in ascending order so that you can clearly see which range any particular value falls into.

In the example above, all data that comes earlier in the alphabet than `Archive` is placed in the first partition. All data whose alphabetical order is the same or later than `Archive` but earlier than `Distribution` is placed in the second partition. All data whose alphabetical order is the same or later than `Distribution` but earlier than `Inbound` is placed in the third partition. All other data is placed in the forth partition.

Note that it is the partition scheme that assigns a specific partition to the range; you can actually assign different ranges to the same partition.

Also, this partition function applies to all queues in the Business Process. For example, if there is a queue `Begin` it falls into the second range and will be assigned to the same partition as the `Archive` queue. But if `Begin` is not a Type 1 queue this result may be less than ideal. One way to ensure that every queue is assigned to its intended partition is to list all the queues in the Business Process in alphabetical order in the partitioning function, and then specify the appropriate partition for each queue. If a new queue is added to the Business Process, you can alter the partition function and partition scheme to take account of this new queue.

Partitioning Scheme

The partitioning scheme uses the partitioning function to define which records (with a particular value of the partitioning function) go to which partition.

```
CREATE PARTITION SCHEME [QNamePScheme]
AS PARTITION [QNamePFN] TO ([PRIMARY], [P1], [P2], [P3])
GO
```

Since our partitioning function is based solely on the value of the 'queue' field, our partitioning scheme tells the database which queue goes to which partition.

Partitioning the Table

To partition the table, simply specify the partitioning scheme for the table:

```
create table interactions
(
id varchar(16) not null,
...
) on QNamePScheme(queue)
Go
```

Note that we explicitly specify that the queue field value should be given to the partitioning scheme and subsequently to the partitioning function to decide which partition the record should go to.

Verification

The following SQL statement is an easy way to monitor how many records are stored in each partition for a given partitioned table (the `interactions` table in this example):

```
SELECT
p.partition_number, fg.name, p.rows
FROM
sys.partitions p
INNER JOIN sys.allocation_units au
ON au.container_id = p.hobt_id
INNER JOIN sys.filegroups fg
ON fg.data_space_id = au.data_space_id
WHERE
p.object_id = OBJECT_ID('interactions')
```

This produces results similar to those shown in the following table.

Partition_number	Name	Rows
1	PRIMARY	1
2	P1	1
3	P2	1
4	P3	1

The table shows that each partition contains a single record. If you insert a new record and execute the above statement again, it will show which partition the new record has been placed in, verifying your partition function and scheme.

> **Important**
>
> To compare performance of the partitioned database with an unpartitioned database, you will need to artificially create a certain distribution of interactions between partitions (different queues) and see how fast the same SELECTs are being executed. When interactions change queues, the records are physically relocated into different partitions (according to the partition scheme).

Converting to and From BLOB

Interaction Server ordinarily stores attached data in the `flexible_properties` field as a BLOB (binary large object).

Converting from BLOB

You can convert attached data to a custom field by running Interaction Server in a special utility mode, in which Interaction Server uses the key-value format of this attached data to convert all such fields to custom fields.

To run Interaction Server in utility mode, launch it from a command line with the following option:

```
-convert-fields [command_or_parameters]
```

where the optional `command_or_parameters` is one of the following:

- `reset`—Ensures that the next run in utility mode will start processing from the beginning, rather than picking up where it left off.
- `bulk-size=N`—Determines the number of records that are processed before committing the transaction. The default value is 100, valid values are any integer in the range 1–1000.

Here is an example command line:

```
interaction_server -host genesys_host -port 9876 -app IxnSrv05 -convert-fields reset
```

You can also have Interaction Server convert an existing database field into a BLOB, stored in the `flexible_properties` field. To do so, use the following procedure.

Converting a field to a BLOB

1. Open the corresponding Business Attribute Value in Configuration Manager.
2. In the `translation` section, add an option called `to-delete` and give it the value `yes`.
3. Run Interaction Server in utility mode, as described above. Interaction Server, in utility mode, moves the content of all such fields into the `flexible_properties` field and leaves the custom field with an empty value.

> **Important**
>
> When Interaction Server runs in utility mode all of its other features are disabled: it cannot process interactions or open ports for clients.

Event Logger

Interaction Server includes Event Logger, a mechanism for storing reporting event messages in a database. You can configure it to store all reporting events or a selected subset. You can also create multiple instances of it.

Interaction Server generates, to registered reporting engines, messages that provide a detailed picture of the processing of each interaction. It classifies these messages, in two ways. The attributes of these messages include much information about the interaction itself, such as its type, time received, associated agents, queues and workbins it was placed it, and so on. For a reference listing of these events and their attributes, see the **Genesyslab.Platform.OpenMedia.Protocols.InteractionServer.Events** in the Platform SDK API Reference for .NET (or Java).

All configuration for the logger functionality is done in the Database Access Point (DAP) associated with the logger database.

There are ways to manage the flow of data produced by Event Logger.

You can have Event Logger send events to a message queue.

Deploying Event Logger

1. Create a database to store the reporting data.
2. Locate the correct setup script for your RDMBS and run it on the database you created in Step 1.

   This script is called `eldb_<database_name>.sql`, where <database_name> is either *db2, mssql,* or *oracle* (for example, `eldb_mssql.sql`). To locate the script, go to the Script subdirectory of the installation directory of your Interaction Server, then open the subdirectory named after your RDBMS; for example, `\InteractionServer_801\Script\Oracle.`
3. Create a Database Access Point (DAP), filling in the usual mandatory settings on the `General` and `DB Info` tabs.
4. On the DAP's `Options` tab, create a section called `logger-settings`. This is the only mandatory section; its existence tells Interaction Server to use this DAP for storing reporting events.
5. In the `logger-settings` section, add at least one option (the section must contain at least one option in order to be valid).
6. Optionally add any of the following section types:
7. `event-filtering`—Contains options filtering out certain classes of event messages
8. `custom-events`—Specifies a custom mapping of the CustomEventId attribute value of `EventCustomReporting` (the option name) to the Event Logger table to store them in (the option values)
9. Custom data `sections`—Five sections that enable you to map the name of any event onto a custom field in the Logger database.
10. On Interaction Server's `Connections` tab, add a connection to the DAP. For multiple instances of the Event Logger, run the creation script multiple times, creating multiple databases. Also create a DAP for each database.

`

Managing Event Logger Data

For the rpt_interaction, rpt_agent, and rpt_esp tables, Genesys supplies a set of scripts that deletes events as soon as processing of the interaction stops, the agent logs out, or the external service responds, respectively. For custom reporting events that are stored in the rpt_custom table, the event-driven trigger trg_del_cust_delay purges them from the rpt_custom table, with a configurable delay (the default is 10 minutes).

If you want to preserve this data, you can disable the triggers trg_delete_stopped, trg_delete_resp, trg_del_cust_delay, and trg_delete_logout after you run the setup script. For Oracle, additionally, disable the triggers trg_mark_cust_logged, trg_mark_responded, trg_mark_ended_session and trg_mark_stopped_ixn.

You can reenable the triggers any time and resume removing records from the database automatically.

Of course event messages increase rapidly in number as interactions are processed, so you will want to take measures to periodically delete data from the database or move it elsewhere.

Also note that after creating or removing custom fields in a database, some triggers become invalid. If this happens, you must recompile them to be sure they work properly.

Classification of Events in Event Logger

The logger functionality classifies reporting events in two ways:

- By activity type—that is, whether the activity refers to an interaction, an agent, an ESP server, or is of a custom type. The database contains tables for each type: interaction activity is stored in `rpt_interaction`, agent activity is stored in `rpt_agent`, and ESP server activity is stored in `rpt_esp`. Custom activity can be stored in `rpt_interaction`, `rpt_agent`, or `rpt_custom`, depending on the configuration in the `custom-events` section of the Event Logger DAP.
- By endpoint type—that is, whether that interaction is being transmitted to a queue, strategy, agent, or ESP service. You can filter out events according to endpoint type. A few events do not have an endpoint type; you cannot filter these events.

The following table lists the events and their classifications.

Event	Activity	Endpoint
EventPropertiesChanged	Interaction	-
EventPartyAdded	Interaction	Agent, Strategy
EventPartyRemoved	Interaction	Agent, Strategy
EventRevoked	Interaction	Agent
EventInteractionSubmited	Interaction	-
EventProcessingStopped	Interaction	-
EventHeld	Interaction	-
EventResumed	Interaction	-
EventPlacedInQueue	Interaction	Queue
EventPlacedInWorkbin	Interaction	Queue
EventAgentInvited	Interaction	Agent
EventRejected	Interaction	Agent
EventTakenFromQueue	Interaction	Queue
EventTakenFromWorkbin	Interaction	Queue
EventAgentLogin	Agent	Agent State
EventAgentLogout	Agent	Agent State
EventDoNotDisturbOn	Agent	Agent State
EventDoNotDisturbOff	Agent	Agent State
EventMediaAdded	Agent	Agent State
EventMediaRemoved	Agent	Agent State
EventNotReadyForMedia	Agent	Agent State
EventReadyForMedia	Agent	Agent State
EventAgentStateReasonChanged	Agent	Agent State
EventMediaStateReasonChanged	Agent	Agent State

Event	Activity	Endpoint
EventExternalServiceRequested	ESP Server	ESP Server
EventExternalServiceResponded	ESP Server	ESP Server
EventCustomReporting	Interaction, Agent, or Custom	-

Event Logger Options

This section provides short descriptions of the DAP options that configure the Event Logger's behavior. See the eServices 8.1 Reference Manual for full details.

logger-settings Section

`batch-size`—Defines the minimum number of records to store in internal memory before flushing to the database. Valid values are 1–5,000; the default is 500.

`max-queue-size`—Defines the maximum number of records that are kept in memory while waiting to be written to the database. If the number of records exceeds this maximum, the data are discarded from memory and are not written to the database. Valid values are 10,000–100,000; the default is 20,000.

`storing-timeout`—Defines a time interval, in milliseconds, between operations of writing to the database. Valid values are 500–60,000; the default is 1,000.

> **Important**
>
> `storing-timeout` and `batch-size` define limits that trigger writing to the database: writing takes place as soon as one or the other is reached.

`schema-name`—Specifies the name of the schema used to access the database.

custom-events Section

This section contains options that list custom events by their identifiers and specify which table (interaction, agent or custom) stores them.

event-filtering Section

This section contains seven options, six of which are named for one of the endpoint types that is referred to in the classification of events:

```
log-agent-state
log-agent-activity
```

```
log-queue
log-strategy
log-esp-service
```

With the value `false`, events associated with the named endpoint type are filtered out. For example, setting `log-queue` to a value of `false` prevents the events `EventPlacedInQueue`, `EventPlacedInWorkbin`, `EventTakenFromQueue`, and `EventTakenFromWorkbin` from being stored. The remaining two options in this section are:

- `log-userdata`—With the value `false`, data from custom fields is filtered out.
- `event-filter-by-id`—A list of comma-separated event identifiers. Only these events are stored in Event Logger. If this option is not present or contains no event identifiers, event filtering by identifier is not applied.

These event identifiers are listed in the Platform SDK 8.1.4 API Reference for .NET (or Java). For example, the identifier of EventRejected is 168.

Custom Data Sections

The five sections contain options specifying a list of events that are to be stored in custom fields of the event logger database. All five work identically, the differences being (a) the events from which the user data is taken and (b) the database table that stores them. These differences are shown in the following table.

Section	Source Event	Logger Database Table
`itx-custom-data`	`UserData` and `EventContent` attributes of interaction-related reporting events	`rpt_interaction`
`esp-custom-data`	`UserData` attribute of `EventExternalServiceRequested` and `EventExternalServiceResponded`	`rpt_esp`
`esp-service-data`	`Envelope3rdServer` attribute of `EventExternalServiceRequested` and `EventExternalServiceResponded`	`rpt_esp`
`agent-custom-data`	`EventContent` attribute of `EventCustomReporting`	`rpt_agent`
`custom-customdata`	`EventContent` attribute of `EventCustomReporting`	`rpt_custom`

For an explanation of the `Envelope3rdServer` attribute, see Platform SDK 8.1.4 API Reference for .NET (or Java).

To use these options, you must first add a field to the appropriate Event Logger database table. Its data type must be the same as that of the mapped user data key. In these sections, the options have the following characteristics:

- The name is a user data key name (case-sensitive).
- The value is three semicolon-separated strings, which specify the following:
  1. The name of the field that you added to the database table. This value is required.
  2. The data type: string, integer, or timestamp. The default is string, with default length 64. If your data type is other than string, or if it is string and you want to specify a non-default length (next item), this value is required.
  3. Optionally, the length. The default for the string type is 64. There are no default values for integer and timestamp.

For example, if you have a data key called `CustomerSegment`, you can add a custom field to store this data as follows:

1. Add a field called `customer_segment` to the `rpt_interaction` table.
2. In the itx-custom-data section, create an option called `CustomerSegment`.
3. Give it this value: `customer_segment;string;64`.

Since string and 64 are the default values for type and length respectively, the value of this option could also be simply `customer_segment`.

Using a Message Queue with Event Logger

You can have Event Logger send events to a message queue, such as IBM MQ-Series, or Microsoft Message Queue (MSMQ). This provides a mechanism for reliable reporting events delivery to Interaction Server's reporting clients. Disconnection of the client does not lead to a loss of reporting events. Instead, events are stored in the message queue and delivered to the client (or otherwise read by the client) after it reconnects.

To use this functionality, you must create a DAP object that is specifically for the streaming of reporting events into MSMQ or MQ-Series. Both Interaction Server and the client connect to this DAP. This DAP must have the following section and options, which partly resemble the sections and options of the Event Logger DAP and are also documented in the eServices 8.1 Reference Manual:

- `logger-settings` section
  - `delivery-protocol`. Possible values are:
    - `event-log`—The default, for using Event Logger database scripts
    - `mq-series`—For the MQ-Series message queue system
    - `msmq`—For the MSMQ message queue system
    - `jms`—For a JMS queue
  - `delivery-queue-name`—The name of the queue to send messages to.

Optionally, Event Logger DAP can use the following section and option:

- `event-filtering` section. Contains the single option `event-filter-by-id`, whose value is a list of comma-separated event identifiers. Only these events are sent to the message queue; events not listed are not sent. This option is analogous to the option of the same name used in the Event Logger DAP.

> **Important**
>
> The event identifiers used in `event-filter-by-id` are listed in the Platform SDK 8.1 API Reference for .NET (or Java).

Using the JMS Event Logger with Apache ActiveMQ

> **Important**
> You must have Apache ActiveMQ 5.14.5 or higher.

You can use the JMS Event Logger with Apache ActiveMQ to process reporting events using a JMS queue.

To enable the JMS Event Logger with Apache ActiveMQ, edit the option **delivery-protocol** and set the value to `jms`.

Configuring JMS Event Logger with Apache ActiveMQ

1. Configure Apache ActiveMQ, as described on the Apache website.
2. Create a **jndi.properties** file with the following content:

```
java.naming.factory.initial = org.apache.activemq.jndi.ActiveMQInitialContextFactory
java.naming.provider.url = tcp://activemq_host:61616
connectionFactoryNames = ConnectionFactory
queue.InxEventLogQueue = InxEventLogQueue
queue.inbound = inx.inbound
queue.error = inx.error
queue.processed = inx.processed
queue.notification = inx.notification
```

> **Important**
> The value of **queue.InxEventLogQueue** refers to a queue name in your environment.

3. Pack the **jndi.properties** file in **amq-jndi.jar**.
4. Add the following jars to the **-Djava.class.path** option in the **jvm-options** section:
   - activemq-all-5.14.5.jar
   - amq-jndi.jar
5. Create the JMS Event Logger as a Database Access Point (DAP).
6. Configure JMS Event Logger for connecting to Apache ActiveMQ by adding the following options in the **logger-settings** section:

- `delivery-protocol=jms`
- `delivery-queue-name=InxEventLogQueue`
- `jms-connection-factory-lookup-name=ConnectionFactory`
- `jms-initial-context-factory=org.apache.activemq.jndi.ActiveMQInitialContextFactory`
- `jms-provider-url=tcp://activemq_host:61616`
- `reconnect-timeout=10`

> **Important**
>
> - Change the value of **InxEventLogQueue** to the queue value you set in the **jndi.properties** file.
> - The value of **jms-provider-url** is the Apache ActiveMQ URL that corresponds to the *<transportConnectorname>* node from the **activemq.xml** file.

7. Configure persistence of messages generated by JMS Event Logger by setting the option recoverable. If this option is set to **true**, the message producer delivery mode is set to **DeliveryMode.PERSISTENT**. Otherwise, if set to **false**, the delivery mode is set to **DeliveryMode.NON_PERSISTENT**.

> **Important**
>
> If the delivery mode is **DeliveryMode.NON_PERSISTENT** and the corresponding message queue is deleted on the fly, Interaction Server does not report any errors, even though the messages are not written anywhere.

8. Optionally, you can repeat the steps in this section to set up multiple JMS Event Loggers. With each additional logger, you must increment the name of additional queues as `queue.InxEventLogQueue2 = InxEventLogQueue2` in the **jndi.properties** file.
9. Add created JMS Event Loggers to Interaction Server connections.

Using TLS with Apache ActiveMQ

1. Prepare the TLS certificates, as described in the Genesys Security Deployment Guide.
2. Copy **cert.jks** and **truststore.jks** into the following folder: ***<Apache ActiveMQ installation directory>*/conf**.
3. Open the file **activemq.xml** in the folder ***<Apache ActiveMQ installation directory>*/conf** and add the following lines:

```
        <transportConnectors>
          ...

<transportConnectorname="ssl"uri="ssl://0.0.0.0:61617?trace=true&needClientAuth=true"/>
          ...
```

```
</transportConnectors>
<sslContext>
    <sslContextkeyStore="file:${activemq.base}/conf/cert.jks"
    keyStorePassword="YourKeyStorePassword"
    trustStore="file:${activemq.base}/conf/truststore.jks"
    trustStorePassword="YourTrustStorePassword"/>
</sslContext>
```

> **Important**
>
> Change the values of **keyStorePassword** and **trustStorePassword** to acceptable passwords.

4. Restart ActiveMQ.
5. Update the following configuration options in the **logger-settings** section:
   - jms-initial-context-factory=org.apache.activemq.jndi.ActiveMQSslInitialContextFactory
   - jms-provider-url=ssl://activemq_host:61617
6. Add the following configuration options in the **jms-additional-context-attributes** section:
   - `connection.ConnectionFactory.keyStore=cert.jks`
   - `connection.ConnectionFactory.keyStorePassword=keyStorePassword`
   - `connection.ConnectionFactory.keyStoreType=jks`
   - `connection.ConnectionFactory.trustStore=truststore.jks`
   - `connection.ConnectionFactory.trustStorePassword=trustStorePassword`
   - `connection.ConnectionFactory.trustStoreType=jks`

> **Important**
>
> Change the values of **connection.ConnectionFactory.keyStorePassword** and **ConnectionFactory.trustStorePassword** to the values you set in the **activemq.xml** file.