This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Interaction Server Administration Guide

Interaction Management 9.0.x

2/14/2022

Table of Contents

Interaction Server Administration Guide

This document 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.
- 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 Clusters

Starting with release 8.5.106.x of Interaction Server and 8.5.107.x of Interaction Server Proxy, you can configure multiple Interaction Servers into a cluster that works with a single instance of Interaction Server Proxy.

KPI Counters

Starting with the 8.5.102.02 release, Interaction Server includes KPI (Key Performance Indicator) counters that monitor:

- The number of requests of different types received from clients.
- The number of Interaction Server protocol errors, counted per error type, sent to clients.
- The number of ESP (External Service Protocol) errors, counted per original request type, sent to clients.

Interaction Server clients can access these counters using `EventPing`.

By default, Interaction Server does not calculate these counters. To enable the counters, set the following options in the **[settings]** section to `true`:

- For requests received, `collect-request-counters`
- For Interaction Server protocol errors, `collect-error-counters`
- For ESP errors, `collect-esp-error-counters`

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.
- Interactions Server dropped support for DB Server in the 8.5.3 release. Therefore, the previous known limitations related to DB Server are no longer applicable.
- 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.
- Take these precautions when configuring Interaction Server for high availability (HA).
- The interaction state timeouts set by the following options are not exact and can be delayed for up to 10 seconds due to the way they are implemented.
  - delivering-timeout
  - handling-timeout
  - routing-timeout
- You cannot use commas (,) and semicolons (;) in interaction queue names.

Interaction Ordering in Clusters

For the ordering of interactions to work correctly in **RequestGetWorkbinContent, RequestFindInteractions** and **RequestTakeSnaphot** in cluster environments, the following conditions must be met:

- All Interaction Server nodes in the cluster must use the same type of database.
- The database encoding and collation must be the same for all the databases used by Interaction Server.
- The encoding used by all the cluster nodes must be the same. Genesys recommends using UTF-8 for Interaction Server. (Interaction Server Proxy neither requires nor allows setting UTF-8 specifically).
- The ordering may contain only predefined interaction properties (a list is provided below) and defined interaction custom properties. The field names may also be used interchangeably with the corresponding property names.

In addition, these conditions are important because the Business Process definition (view conditions and orders) may be specific to the particular database type, and all nodes within the cluster use the same business process.

Below is a list of the field names and corresponding predefined interaction property names that can be used in the attributes that specify interaction order:

Field name	Property name
abandoned_at	AbandonedAt
assigned_at	AssignedAt
assigned_to	AssignedTo
completed_at	CompletedAt
delivered_at	DeliveredAt
external_id	ExternalId
held_at	HeldAt
id	InteractionId
is_locked	IsLocked
is_online	IsOnline
media_type	MediaType
moved_to_queue_at	MovedToQueueAt
parent_id	ParentId
place_in_queue_seq	PlaceInQueueSeq
placed_in_queue_at	PlacedInQueueAt
priority	Priority
queue	Queue
received_at	ReceivedAt
scheduled_at	ScheduledAt
service_objective	ServiceObjective
state	InteractionState
submit_seq	SubmitSeq
submitted_at	SubmittedAt
submitted_by	SubmittedBy
subtype	InteractionSubtype
tenant_id	TenantId
type	InteractionType
workbin	Workbin

Limitations

The following limitations apply to **RequestGetWorkbinContent** ordering:

- The **max-workbin-content** option in Interaction Server Proxy must be less than or equal to the minimum possible value of the same option among all nodes.

- Ordering conditions in views related to workbin queues must contain valid interaction field/property names.
- Functions (for example, `getutdata()` or `_length`) in views related to workbin queues are not supported. Otherwise, sorting occurs by `received_at, id`.

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

In release 7.6.1 and later, Interaction Server includes Event Logger, a mechanism for storing reporting event messages in a database or a message queue. 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.

Deploying Event Logger

This page tells you how to deploy event logger.

Prerequisites for ODBC 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 `postgre`, `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`.

Create Event Logger DAP

1. Create a Database Access Point (DAP), filling in the usual mandatory settings on the `General` and `DB Info` tabs.
   - For ODBC Event Logger, provide the correct information about the database. Refer to Configuring Interaction Server DAP for information how to configure connection to DB correctly.
   - For other Event Loggers DB information is not used and can be filled with any values.
2. 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 logging reporting events.
3. Specify one of the following values depending on the Event Logger type in the `delivery-protocol` option in `logger-settings`:
   - `odbc` — 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
   - `groovy` — For the Groovy Event Logger
4. Optionally, in the `logger-settings` section, add any other option.
5. Optionally add any of the following section types:
   - `event-filtering` — Contains options filtering out certain classes of event messages
   - `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)
   - Custom data `sections` — Five sections that enable you to map the name of any event onto a custom field in the Logger database.

6. On Interaction Server's `Connections` tab, add a connection to the DAP.

For multiple instances of the Event Logger, repeat this procedure multiple times.

Important
Each instance of ODBC Event Logger requires separate database so multiple databases must be created if multiple ODBC Event Loggers are used.

Preparing DAP configuration objects

New environment

For a new environment, create a Database Access Point (DAP) object as described in the Framework Database Connectivity Reference Guide and add the DAP to the connections of your Interaction Server.

To configure the database-oriented Event Logger for Interaction Server, create a second DAP object as described on the Deploying Event Logger page.

Existing environment

For an existing environment with Interaction Server 8.5.1 - 8.5.3 and DB Server, you should configure them to use ODBC to connect to database before the upgrade to version 9.0.0. Use Genesys Administration Extension (GAX) to create copies of the existing DAP objects used by Interaction Server. You can use these copies to switch back to connecting via DB Server while testing.

Configure the DAP

This section applies to Interaction Server and Event Logger. Related information about Database Capture Point is in a separate location.

For an ODBC connection, you must configure the Database Access Point (DAP) associated with the Application in question, as follows:

1. For Database Info configuration, in Genesys Administrator, enter the following in the **[DB Info]** section of the **Configuration** tab (in Configuration Manager, on the **DB Info** tab):
   - DBMS Name - host name of the database server
   - DBMS Type - type of the database
   - Database Name - name of the database
   - User Name - user name to use to connect to the database
   - Password - password for the user name account
2. Configure the options of Database Access Point application to use ODBC.

- For Interaction Server database DAP
  ```
  [settings]

  dbprotocol=odbc

  connection-string = <your connection string>
  ```
- For Database Event Logger DAP
  ```
  [logger-settings]

  delivery-protocol =odbc

  connection-string = <your connection string>
  ```

3. Determine the Connection string value.

Configure ODBC driver in connection string

To make configuration easier, Interaction Server will try to build the ODBC connection strings itself based on the information that is available in the DAP. During this process, Interaction Server uses the following default names for the drivers:

Database Type	Default Driver
MS SQL	{SQL Server Native Client 10.0}
Oracle	{Oracle in OraClient11g_home1}
PostgreSQL	{PostgreSQL ANSI(x64)}

Important

Genesys strongly recommends that you use latest versions of ODBC drivers that were released officially for your database. The main purpose of default drivers is backward compatibility with old database.

If your configuration uses a different driver name, you must explicitly provide the actual driver name in value of **connections-string** option in section **[settings]** (or **[logger-settings]** for the Event Logger DAP).

The value of **connection-string** is a list of key-value pairs separated by semicolons (;).

- To specify the actual driver, give it a value of `driver=<driver_name>`.
- If your database server is not running on the default port (1433 for MSSQL and 1521 for Oracle), you must also provide the port number by adding port=<actual_port> pair to the the value of connection-string. For example, *Driver=SQL Server Native Client 11.0;port=1433*.

Important

For Oracle, ensure the L0B=T option is present in the connection string if it’s not specified in DSN (see the next section).

DSN

You can also have Interaction Server use a DSN that has been configured in your system. To do this, locate or create the **connection-string** option in the **[settings]** section of the DAP (the **[logger-settings]** section in the Event Logger DAP), and set it to `DSN=<name_of_the_dsn>`. With this method, the user name and password to connect to the database are taken from the settings on the **DB Info** tab and do not need to be provided in the DSN properties.

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

`delivery-protocol`—Defines type of the event logger. Possible values are:

- `odbc` — 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
- `groovy` — For the Groovy Event Logger

> **Important**
> Since Interaction Server 8.5.3 value for this option must be specified always. And since this version value "event-log" is not supported.

`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.

`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.

`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.

`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 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	`rpt_esp`

Section	Source Event	Logger Database Table
	`EventExternalServiceRequested` and `EventExternalServiceResponded`	
`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 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`.

Preparing DAP configuration objects

New environment

For a new environment, create a Database Access Point (DAP) object as described in the Framework Database Connectivity Reference Guide and add the DAP to the connections of your Interaction Server.

To configure the database-oriented Event Logger for Interaction Server, create a second DAP object as described on the Deploying Event Logger page.

Existing environment

For an existing environment with Interaction Server 8.5.1 - 8.5.3 and DB Server, you should configure them to use ODBC to connect to database before the upgrade to version 9.0.0. Use Genesys Administration Extension (GAX) to create copies of the existing DAP objects used by Interaction Server. You can use these copies to switch back to connecting via DB Server while testing.

Configure the DAP

This section applies to Interaction Server and Event Logger. Related information about Database Capture Point is in a separate location.

For an ODBC connection, you must configure the Database Access Point (DAP) associated with the Application in question, as follows:

1. For Database Info configuration, in Genesys Administrator, enter the following in the **[DB Info]** section of the **Configuration** tab (in Configuration Manager, on the **DB Info** tab):
   - DBMS Name - host name of the database server
   - DBMS Type - type of the database
   - Database Name - name of the database
   - User Name - user name to use to connect to the database
   - Password - password for the user name account
2. Configure the options of Database Access Point application to use ODBC.
   - For Interaction Server database DAP
     ```
     [settings]

     dbprotocol=odbc

     connection-string = <your connection string>
     ```
   - For Database Event Logger DAP
     ```
     [logger-settings]

     delivery-protocol =odbc

     connection-string = <your connection string>
     ```
3. Determine the Connection string value.

Configure ODBC driver in connection string

To make configuration easier, Interaction Server will try to build the ODBC connection strings itself based on the information that is available in the DAP. During this process, Interaction Server uses the following default names for the drivers:

Database Type	Default Driver
MS SQL	{SQL Server Native Client 10.0}
Oracle	{Oracle in OraClient11g_home1}
PostgreSQL	{PostgreSQL ANSI(x64)}

Important

Genesys strongly recommends that you use latest versions of ODBC drivers that were released officially for your database. The main purpose of default drivers is backward compatibility with old database.

If your configuration uses a different driver name, you must explicitly provide the actual driver name in value of **connections-string** option in section **[settings]** (or **[logger-settings]** for the Event Logger DAP).

The value of **connection-string** is a list of key-value pairs separated by semicolons (;).

- To specify the actual driver, give it a value of `driver=<driver_name>`.
- If your database server is not running on the default port (1433 for MSSQL and 1521 for Oracle), you must also provide the port number by adding port=<actual_port> pair to the the value of connection-string. For example, *Driver=SQL Server Native Client 11.0;port=1433*.

Important

For Oracle, ensure the `LOB=T` option is present in the connection string if it's not specified in DSN (see the next section).

DSN

You can also have Interaction Server use a DSN that has been configured in your system. To do this, locate or create the **connection-string** option in the **[settings]** section of the DAP (the **[logger-settings]** section in the Event Logger DAP), and set it to `DSN=<name_of_the_dsn>.` With this method, the user name and password to connect to the database are taken from the settings on the **DB Info** tab and do not need to be provided in the DSN properties.

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:
    - `odbc`—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
    - `groovy`—For the Groovy Event Logger
  - 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 9.0.x 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

   jvm-options are located in Interaction Server application options. They are configured on per-process basis and not on per-capture-point one.
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

   - These options are for ActiveMQ credentials. These credentials might be specified during the connection factory configuration procedure.
   - 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.

Using the JMS Event Logger with Oracle OpenMQ

This page provides an example of configuring a JMS Event Logger Application object when using the OpenMQ provider.

Configuring JMS Event Logger Application with Oracle OpenMQ

1. Create Database Access Point Application in the Configuration Manager named `EL_OpenMQ`.
2. On the **Options** tab, create a section named `logger-settings`. In this section add the following options:
    - `[logger-settings]`
    - `delivery-protocol=jms`
    - `jms-connection-factory-lookup-name=ConnectionFactory`
    - `jms-initial-context-factory=com.sun.jndi.fscontext.RefFSContextFactory`
    - `delivery-queue-name= IxnEventLogQueue`
    - `jms-provider-url= file:///D:/OpenMQExample` (the path points to the folder where the **.bindings** file (in UNIX file format) is stored on the Interaction Server host)
3. On the **Connections** tab of the Interaction Server Application definition, add the created Event Logger.
4. Configure the Interaction Server options to load JVM and all of the required libraries. Interaction Server Configuration for working with OpenMQ is described in the OpenMQ—Interaction Server JVM section of the JMS Capture Point Guide.

OpenMQ—JMS EventLogger Queue

This section provides an example of setting up queue for the JMS Event Logger when using OpenMQ provider.

Setting up EventLogger queue with Open Message Queue Administration Console

1. Connect to the OpenMQ broker that is running.
2. Add the following queue using the **Add Broker Destination dialog**: **IxnEventLogQueue**
3. For the queue that you have added, set **Max Number of Producers** and **Max Number of Active Consumers** to **Unlimited**.
4. Add a new Object Store and set the following JNDI Naming Service Properties:

    1. Set **java.naming.factory.initial** to **com.sun.jndi.fscontext.RefFSContextFactory**.
    2. Set **java.naming.provider.url** to **file:///D:/OpenMQExample**.

> **Important**
> This is the directory in which the **.bindings** file containing definitions will be saved.

5. Connect to the newly created object store.
6. Add a connection factory object using the **Add Connection Factory Object** dialog:
    1. Specify the lookup name, such as **ConnectionFactory**.
    2. Specify the **Factory Type** as **QueueConnectionFactory**.
    3. In the **Client Identification** tab, specify the **Default Username** and **Default Password** (for example, `guest/guest`).
7. Add destinations to the object store for all four queues that you defined previously.

> **Important**
> The lookup names can be different from the destination names.

8. After the above steps have been completed, the folder **D:/OpenMQExample/** contains the **.bindings** file with connection factory and queue definitions. Open the file, examine it for the presence of the defined queues and connection factory, and save it with File format set to **UNIX** so that it is possible to use it on UNIX operating systems.

Using the JMS Event Logger with TIBCO EMS

This example assumes the following:

- The host of the TIBCO message queue service is called **tibhost**.
- Queue called **event** is defined.
- Both user name and password are **guest**.
- The connection factory is called **tibconnectionfact**.

Configuring JMS Event Logger Application with TIBCO EMS

1. On the **Options** tab, create a section called **logger-settings**. In this section, add the following options:
    - `delivery-protocol =jms`
    - `delivery-queue-name =event` (the same as the queue name)
    - `username=guest`
    - `password=guest`
    - `jms-connection-factory-lookup-name=tibconnectionfact` (the name of the connection factory on TIBCO)
    - `jms-initial-context-factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory`
    - `jms-provider-url=tibjmsnaming://tibhost:7222`
2. On the **Connections** tab of the Interaction Server which will use Event Logger, add connection to the just created Event Logger.
3. Configure the Interaction Server options to load JVM and all of the required libraries. Interaction Server Configuration for working with TIBCO is described in the TIBCO—Interaction Server JVM section of the JMS Capture Point Guide.

Using the JMS Event Logger with IBM WebSphere MQ

This page provides an example of configuring a JMS Event Logger Application object when using WebSphere MQ.

This is a specific example of the more general configuration procedure.

Configuring JMS Event Logger Application with IBM WebSphere MQ

1. On the **Options** tab, create a section called `logger-settings`. In this section add the following options:
   - `Delivery-protocol=jms`
   - `delivery-queue-name =event`(the same as the corresponding Destination name)
   - `jms-connection-factory-lookup-name=my_ConnFactory` (the name of the connection factory that you created in WebSphere MQ)
   - `jms-initial-context-factory=com.sun.jndi.fscontext.RefFSContextFactory`
   - `jms-provider-url=file:///home/InteractionServer` (the path points to the folder where the **.bindings** file, in UNIX file format, is stored on the Interaction Server host)
2. On the **Connections** tab of the Interaction Server which will use this JMS Message queue, add the just created Event Logger.

Next Steps

Configure the Interaction Server options to load JVM and all of the required libraries. Interaction Server Configuration for working with WebSphere is described in the WebSphereMQ—Interaction Server JVM section of the JMS Capture Point Guide.

WebSphereMQ-JMS Event Logger Queue

This page provides an example of setting up queue for the JMS Event Logger when using IBM WebSphere MQ.

Setting up queue using IBM WebSphere MQ Explorer

1. Start WebSphere MQ Explorer. Find the Object tree in the Navigator window.
2. Right-click the **Queue Managers** node and select **New** to create a new Queue Manager. Follow the

steps in the resulting Wizard, choosing a name (for example, `my_QManager`) and unique listening port.

3. As the Object tree is updated, find the **Queue** node under the new Queue Manager. Right-click this node and select **New** > **Local Queue**.
4. Create Local Queue named **mq_event**. Select **Persistent** for the **Default Persistence** setting.
5. With the **Queue** node selected in the Object tree, right-click **mq_event** in the Content pane and select **Put Test Message**. Enter any text of your choice in the **Message data** field, then click **Put message**. This test message waits in the queue until the Event Logger retrieves it.
6. In the Object tree, right-click the **JMS Administered Objects** node and select **Add Initial Context**. Choose **File system** for the JNDI namespace location and select the directory where the corresponding storage file will be created.
7. The new node for initial context now appears in the Object tree. Select it and verify that the **Connection Factories** and **Destinations** nodes appear under it. If necessary, right-click and use the context menu to connect to the `InitialContext` object make these nodes visible.
8. Right-click **Connection Factories** and select **New** > **Connection Factory**. Enter or select the following values:
   - Sample name—`my_ConnFactory`
   - Messaging provider—`WebSphere MQ`
   - Transport—`MQ Client`
   - Base queue manager and Broker queue manager (last screen)—the Queue Manager that you created in Step 2.
   - Host name and port—Correct values for your environment
9. Right-click **Destinations** and select **New** > **Destination** and create a new Destination that corresponds to the queue that you created in Step 4:
   - Type—`Queue`
   - Names—`jms-event`
   - On the last screen, select the proper **Queue Manager** and **Queue** objects.
10. Find the file named **.bindings** at the location established in Step 6. It will be referred to later on the sample configuration.

Using the Groovy Event Logger

You can use the Groovy Event Logger to process reporting events using custom Groovy scripts.

The Groovy Event Logger is configured, as any other event logger, using the Database Access Point application and has its own specific options.

To enable the Groovy Event Logger, edit the option **delivery-protocol** and set the value to `groovy`.

The Interaction Server IP (installation package) contains sample Groovy projects that implement the following types of event loggers:

- Elasticsearch Event Logger—Logs the reporting events to Elasticsearch.
- Kafka Event Logger—Logs reporting events to the Kafka message queue.
- File Event Logger—Simplified sample that logs the reporting events to files. Only use this project as a starting point or for integration with third-party systems.

The IP also contains a document that describes the sample projects, as well as an XML template file for each sample project.

Using the Groovy Event Logger with Apache Kafka

You can use the Groovy Event Logger to send reporting events to an Apache Kafka topic.

Prerequisites

The following prerequisites must be met in order to enable sending of the reporting events to Apache Kafka via Groovy Event Logger:

- Existing Kafka cluster must be present. It should be running brokers of version “0.10.1.0” or newer.
- Minimum JRE 8 is required.
- OpenJDK 11 is recommended.
- Sending of the reporting events to Apache Kafka is supported by using Groovy Event Logger and Java Kafka client and requires Java in order to work. If Java is not installed or not properly configured, this functionality will not be available. All required jar files are provided in IP. Java should be correctly configured in Interaction Server options. See Java configuration in the Integrated Capture Point Guide.

Configure Interaction Server application object

Add the following jars from the Interaction Server installation folder to the `-Djava.class.path` option in the `jvm-options` section:

- `lib/ixn-java-aux.jar`
- `lib/groovy-all-2.4.21.jar`

Configure Kafka Groovy Event Logger application object

To configure Kafka Groovy Event Logger application object:

1. Import the application template from `GroovyEventLoggerTemplates/KafkaGroovyEventLogger.apd` from the Interaction Server installation folder.
2. Create a new application using the newly imported application template.
3. Fill in the usual mandatory settings on the General and DB Info tabs. Following mandatory settings are not used and can be filled with the arbitrary values like:
    - DBMS Name = `groovy`

- DBMS Type = `DB2`
- Username = `groovy`

4. Verify value of `logger-settings/script-classpath` points to `lib/KafkaEventLogger` in the Interaction Server installation folder.
5. Specify the following options in the new application:
    - `kafka-settings/servers` - comma-separated list of the Kafka broker(s) to connect to initially
    - `kafka-settings/topic` - topic to send reporting events to
6. Specify the following option in the new application if you want to send messages in Genesys binary format rather than in JSON (default)
    - `kafka-settings/binary-output=true`
7. On Interaction Server's **Connections** tab, add a connection to the DAP.

Advanced Java Kafka Client Options

Java Kafka client used for communication with the Kafka cluster has many configurable options. Options specified in the `producer-options` section are passed to the Java Kafka producer as is. Producer is used to send message to the Kafka cluster. The list of options can be found on the official page. Any of these options can be configured in this section except `bootstrap.server`. Value of `kafka-settings/servers` is used for the latter option.