SIEMENS

Edge Streaming Analytics - Connector Reference

System Manual

02/2020
V1801.Feb/2020.1

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

⚠ DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

⚠ WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

⚠ CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

⚠ WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens AG
Division Digital Factory
Postfach 48 48
90026 NÜRNBERG
GERMANY

V1801.Feb/2020.1
℗ 02/2020 Subject to change

Table of contents

Document history

1

Version	Date	Changes	Link
V1801.Feb/2020.1	2020-02-22	New document	-

Overview

2

2.1 General Overview

Connectors and adapters enable you to stream data into or out of an edge stream processing engine. They use the publish/subscribe API to interface directly with a variety of message buses, communication fabrics, drivers, and clients.

Connectors are instantiated in the same process space as the edge stream processing engine. A single instance of a connector reads from or writes to a single window of an edge stream processing model.

Adapters are stand-alone executable files, usually built from connector classes. Thus, some adapters are executable versions of their corresponding connector.

Table 2-1 Connectors and Adapters

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
Adapter (Page 17)	Functions as a wrapper around an adapter, enabling the Edge Streaming Server to process the adapter as a connector.	Yes	Yes	No
BACNet (Page 46)	Enables you to stream data over a communications protocol for Building Automation and Control (BAC) networks. The protocol is based on the ISO 16484-5 standard protocol.	Yes	Yes	Yes
BoardReader (Page 164)	Enables you to stream data from a feed aggregator. The site boardreader.com enables you to search message boards, websites, blogs, and other social media.	No	Yes	No

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
SAS Cloud Analytic Services (Page 144)	Enables you to stream data to and from SAS Cloud Analytic Services (CAS). The CAS server is suitable for both on- premises and cloud deployments. It provides the run-time environment for data management and analytics.	No	Yes	No
Cassandra (Page 140)	Enables you to stream data to and from Apache Cassandra databases.	No	Yes	No
Database	Support publish and subscribe operations to a large number of databases.	Yes	Yes	No
Event Stream Processor (Page 129)	Enables you to subscribe to a window and publish what it passes into another Source window.	No	Yes	Yes
File and Socket (Page 116)	Support both publish and subscribe operations on files or socket connections that stream a large number of data types.	Yes	Yes	Yes
HDFS (Page 150)	Supports publish and subscribe operations to a Hadoop Distributed File System.	No	Yes	No
Java Message Service (JMS) (Page 166)	Bundles the JMS publisher and subscriber clients. JMS is a Java Message Oriented Middleware (MOM) API to send messages between two or more clients.	No	Yes	No

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
Kafka (Page 72)	Enable you to stream data to and from an open-source streaming platform developed by the Apache Software Foundation. The platform is engineered to publish and subscribe to streams of records, similar to a message queue or an enterprise messaging system.	Yes	Yes	Yes
Kinesis (Page 87)	Enable you to use Amazon Kinesis Data Streams to collect and process streams of data records in real time.	Yes	Yes	Yes
Modbus (Page 41)	Enable you to stream data to and from Modbus, a serial communications protocol commonly used to connect industrial electronic devices.	Yes	Yes	Yes (Disabled)
MQTT (Page 23)	Enable you to stream data through MQ Telemetry Transport (MQTT), a simple and lightweight publish/subscribe messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks.	Yes	Yes	Yes
OPC-UA (Page 35)	Enable you to stream data through OPC Unified Architecture (OPC-UA), a machine-to-machine communication protocol for industrial automation. OPC-UA is used for communication with industrial equipment and systems for data collection and control.	Yes	Yes	Yes

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
PI (Page 201)	Enable you to stream data to and from a PI Asset Framework server. This server is a repository for asset-centric models, hierarchies, objects, and equipment.	Yes	Yes	Yes
Project Publish (Page 129)	Enables you to subscribe to event blocks that are produced by a window from a different project within the edge stream processing model.	Yes	Yes	Yes
Pylon (Page 109)	Communicate with a Basler GigE camera to continuously publish captured frames into a Source window. The camera must have a known, fixed IP address, and the attached Ethernet network cable must provide power using Power-over- Ethernet.	Yes	Yes (Disabled)	Yes

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
Rabbit MQ (Page 51)	Enable you to stream data to and from RabbitMQ, a lightweight open-source message broker.	Yes	Yes	Yes
REST (Page 182)	Supports subscribe operations to generate HTTP POST requests to a configured REST service.	No	Yes	No
SMTP (Page 179)	Enable you to stream data through the Simple Mail Transfer Protocol, an internet standard for electronic mail (email) transmission.	Yes	Yes	No

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
Sniffer (Page 185)	Enable you to stream data from a packet sniffer. A sniffer is a software program or hardware device that intercepts and logs traffic that passes over a digital network or part of a network.	Yes	Yes	No
Solace (Page 63)	Enable you to stream data to and from Solace, a message broker to establish event-driven interactions between applications and microservices.	Yes	Yes	Yes
Teradata (Page 131)	Enable you to stream data to and from a Teradata server.	Yes	Yes	No
Teradata Listener (Page 136)	Enable you to stream data from a Teradata Listener, which ingests and distributes extremely fast moving data streams throughout an analytical ecosystem.	Yes	Yes	No
Tervela (Page 192)	Enable you to stream data to and from Tervela. Tervela is a data fabric that enables you to capture, share, and distribute data from a number of enterprise and cloud data sources.	Yes	Yes	No
Tibco Rendezvous (Page 96)	Enable you to stream data to and from Tibco RV, a software product that provides a message bus for enterprise application integration (EAI).	Yes	Yes	Yes (Disabled)
Timer (Page 106)	Generate and publish trigger events at regular intervals.	Yes	Yes	Yes
Twitter (Page 155)	Consumes Twitter streams and injects event blocks into Source windows of an engine.	No	Yes	No

Name	Description	Connector Available?	Adapter Available?	Included in ESS?
Twitter Gnip (Page 162)	Consumes data from a Twitter Gnip firehose stream. The adapter is a Python script that invokes the Edge Streaming Analytics C publish/subscribe and JSON libraries.	No	Yes	No
URL (Page 105)	Consumes data into a project through a URL-based connection.	Yes	No	Yes
UVC (Page 112)	Enable you to publish photos taken by a V4L2–compatible camera to an edge stream processing model.	Yes	Yes	Yes
WebSocket (Page 190)	Enables you to read data over a WebSocket-based connection and publish the data into a project.	Yes	No	No
IBM WebSphere (Page 171)	Support the IBM WebSphere Message Queue Interface for publish and subscribe operations.	Yes	Yes	No

2.2 Connectors

2.2.1 Overview to Connectors

Connectors use the Edge Streaming Analytics publish/subscribe API to do one of the following:

- publish event streams into Source windows. Publish operations do the following, usually continuously:
  - read event data from a specified source
  - inject that event data into a specific Source window of a running event stream processor
- subscribe to window event streams. Subscribe operations write output events from a window of a running event stream processor to the specified target (usually continuously).

Connectors do not simultaneously publish and subscribe.

You can find connectors in libraries that are located at **$DFESP_HOME/lib/plugins**. On Microsoft Windows platforms, you can find them at **%DFESP_HOME%\bin\plugins**.

All connector classes are derived from a base connector class that is included in a connector library. The base connector library includes a connector manager that is responsible for loading connectors during initialization. This library is located in **$DFESP_HOME/lib/**

libdfxesp_connectors-Maj.Min, where `Maj.Min` indicates the release number for the distribution.

Excluding Connectors

The **$DFESP_HOME/lib/plugins** directory contains the complete set of plug-in objects supported by Edge Streaming Analytics. Plug-ins that contain "`_cpi`" in their filename are connectors.

When the connector manager starts, all connectors that are found in the **plugins** directory are loaded except those specified as excluded in **esp-properties.yml**. This file is located in the configuration directory.

```
connectors:
excluded:
mq: true
tibrv: true
sol: true
tva: true
pi: true
tdata: true
rmq: true
sniffer: true
mqtt: true
pylon: true
modbus: true
```

By default, excluded connectors require third-party libraries that are not shipped with Edge Streaming Analytics. Because listed connectors are not automatically loaded, errors due to missing dependencies are prevented.

Orchestrating Connectors

Connector orchestration enables you to define the order in which connectors within a project execute, depending on the state of the connector. This enables you to create self-contained projects that orchestrate all of their inputs. Connector orchestration can be useful to load reference data, inject bulk data into a window before injecting streaming data, or with join windows.

By default, all connectors start automatically when their associated project starts and they run concurrently. After you enable connector orchestration, only orchestrated connectors are started when their project starts.

You can represent connector orchestration as a directed graph, similar to how you represent a continuous query. In this case, the nodes of the graph are connector groups, and the edges indicate the order in which groups execute.

Connectors ordinarily are in one of three states: stopped, running, or finished. Subscriber connectors and publisher connectors that are able to publish indefinitely (for example, from a message bus) never reach finished state.

In order for connector execution to be dependent on the state of another connector, both connectors must be defined in different connector groups. Groups can contain multiple connectors, and all dependencies are defined in terms of the group, not the individual connectors.

When you add a connector to a group, you must specify a corresponding connector state as well. This state defines the target state for that connector within the group. When all connectors in a group reach their target state, all other groups dependent on that group are satisfied. When a group becomes satisfied, all connectors within that group enter running state.

Consider the following configuration that consists of four connector groups: G1, G2, G3, and G4:

```
G1: {<connector_pub_A, FINISHED>, <connector_sub_B, RUNNING>}
G2: {<connector_pub_C, FINISHED>}
G3: {<connector_pub_D, RUNNING>}
G4: {<connector_sub_E, RUNNING>}
```

Now consider the following orchestration:

- G1 -> G3: start the connectors in G3 after all of the connectors in G1 reach their target states.
- G2 -> G3: start the connectors in G3 after all of the connectors in G2 reach their target states. Given the previous orchestration, this means that G3 does not start until the connectors in G1 and in G2 reach their target states.
- G2 -> G4: start the connectors in G4 after all of the connectors in G2 reach their target states.

Because G1 and G2 do not have dependencies on other groups, all of the connectors in those groups start right away. The configuration results in the following orchestration:

1. When the project is started, `connector_pub_A`, `connector_sub_B`, and `connector_pub_C` start immediately.
2. When `connector_pub_C` finishes, `connector_sub_E` is started.
3. `connector_pub_D` starts only after all conditions for G3 are met, that is, when conditions for G1 and G2 are satisfied. Thus, it starts only when `connector_pub_A` is finished, `connector_sub_B` is running, and `connector_pub_C` is finished.

A connector group is defined by calling the project `newConnectorGroup`() method, which returns a pointer to a new `dfESPconnectorGroup` instance. If you pass only the group name, the group is not dependent on any other group. Conversely, you can also pass a vector or variable list of group instance pointers. This defines the list of groups that must all become satisfied in order for the new group to run.

After you define a group, you can add connectors to it by calling the `dfESPconnectorGroup::addConnector`() method. This takes a connector instance (returned from `dfESPwindow::getConnector`()), and its target state.

The XML code to define this orchestration is as follows:

```
<connector-group name='G1'>
<connector-entry connector='contQuery_name/window_for_pub_A/pb_A'
state='finished'/>
<connector-entry connector='contQuery_name/window_for_sub_B/sub_B'
state='running'/>
</connector-group>
<connector-group name='G2'>
<connector-entry connector='contQuery_name/window_for_pub_C/pub_C'
state='finished'/>
</connector-group>
<connector-group name='G3'>
```

```
<connector-entry connector='contQuery_name/window_for_pub_D/pub_D'
state='running'/>
</connector-group>
<connector-group name='G4'>
<connector-entry connector='contQuery_name/window_for_sub_E/sub_E'
state='running'/>
</connector-group>
</connector-groups>
<edges>
<edge source='G1' target='G3'/>
<edge source='G2' target='G3'/>
<edge source='G2' target='G4'/>
</edges>
</project-connectors>
```

The corresponding C++ code is as follows:

```
dfESPconnectorGroup*G1=   project->newConnectorGroup("G1");
G1-> addConnector(pub_A, dfESPabsConnector::state_FINISHED); G1->
addConnector(sub_B,
dfESPabsConnector::state_RUNNING);dfESPconnectorGroup*G2=   project-
>newConnectorGroup("G2");
G2-> addConnector(pub_C, dfESPabsConnector::state_FINISHED);
dfESPConnectorGroup*G3= project->newConnectorGroup("G3",2,G1,G2);
G3->
addConnector(pub_D, dfESPabsConnector::state_RUNNING);
dfESPConnectorGroup*G4= project->newConnectorGroup("G4",1,G2); G4->
addConnector(sub_E, dfESPabsconnector::state_RUNNING);
```

2.3 Adapters

2.3.1 Overview to Adapters

Adapters use the publish/subscribe API to do the following:

- publish event streams into an engine
- subscribe to event streams from engine windows

Many adapters are executable versions of connectors. Thus, the required and optional parameters of most adapters directly map to the parameters of the corresponding connector. Unlike connectors, adapters can be networked.

Adapters must have full access to all Edge Streaming Analytics libraries. Therefore, if you run the adapter on a system separate from the streaming engine provided with the Edge Streaming Server installation kit, you must install Edge Streaming Analytics on this system without onboarding it as a MindSphere agent. Thus this system will not use a product license in your tenant.

You can find adapters in **$DFESP_HOME/bin**.

Adapters by Source Language

Adapters are written in C++, Java, or Python. An adapter's source language determines the location of its configuration file, how it uses various communication fabrics, and its command line syntax.

C++ Adapters:

- Bacnet Publisher (Page 46)
- Database
- Event Stream Processor (Page 129)
- File and Socket (Page 116)
- Kafka (Page 72)
- Modbus (Page 41)
- MQTT (Page 23)
- OPC-UA (Page 35)
- PI (Page 201)
- Pylon Publisher (Page 109)
- Rabbit MQ (Page 51)
- SMTP Subscriber (Page 179)
- Sniffer Publisher (Page 185)
- Solace Systems (Page 63)
- TD Listener Subscriber (Page 136)
- Teradata Subscriber (Page 131)
- Tervela Data Fabric (Page 192)
- Tibco Rendezvous (RV) (Page 96)
- Timer Publisher (Page 106)
- UVC Publisher (Page 112)
- IBM WebSphere MQ (Page 171)

Java Adapters:

- SAS Cloud Analytic Server (Page 144)
- Cassandra (Page 140)
- HDAT Reader
- HDFS (Hadoop Distributed File System) (Page 150)
- Java Message Service (JMS) (Page 166)
- REST Subscriber (Page 182)
- Twitter Publisher (Page 155)

Note

The Java adapters are built using Java version 8. You must use Java SE Run-time Environment 8 on any platform running a Java adapter.

Python Adapters:

- BoardReader (Page 164)
- Twitter GNIP (Page 162)

C++ adapters obtain their configuration parameters from the configuration file in the configuration directory. Java adapters use **javaadapters.config**.

Note

Java adapter parameters must be fully specified in the Java adapter configuration file. For a list of the parameter names that must be specified in **javaadapters.config** for each Java adapter, see the documentation for the adapters listed above.

C++ and Java adapters can publish or subscribe using a Rabbit MQ server instead of a direct TCP/IP connection to the Edge Streaming Server. Each adapter has a configuration parameter to tell it to use the Rabbit MQ transport type. When using the Rabbit MQ transport type, an adapter uses a code library to implement the Rabbit MQ protocol. Adapters written in C++ use the rabbitmq-c libraries. Adapters written in Java use dfx-esp- rabbitmq-api.jar and rabbitmq-client.jar. You must obtain rabbitmq-client.jar from the Rabbit MQ Java client libraries at http://www.rabbitmq.com/java-client.html. Install rabbitmq-client.jar in **$DFESP_HOME/lib** on your system.

C++ and Java adapters can publish or subscribe using a Kafka cluster instead of a direct TCP/IP connection to the Edge Streaming Server. Each adapter has a configuration parameter to tell it to use the Kafka transport type. When using the Kafka transport type, an adapter uses a code library to implement the Kafka protocol. Adapters written in C++ use the librdkafka libraries. Adapters written in Java use dfx-esp-kafka-api.jar and kafka-clients-*.jar. You must obtain kafka-clients-*.jar at http://kafka.apache.org/downloads.html. Install kafka-clients-*.jar in

$DFESP_HOME/lib on your system.

All adapters can publish or subscribe using a Solace fabric instead of a direct TCP/IP connection to the Edge Streaming Server. Each adapter has a configuration parameter to tell it to use the Solace transport type. When using the Solace transport type, an adapter uses a code library to implement the Solace protocol. Adapters written in C++ use the libsolclient libraries. Adapters written in Java use dfx-esp-solace-api.jar and sol-*.jar. You must obtain sol-*.jar from Solace Systems. Install sol-*.jar in **$DFESP_HOME/lib** on your system.

Command Line Syntax for C++ and Java Adapters

To run a C++ or Java adapter on the command line, the general syntax is as follows:

```
dfesp_adapter_adapter -Ckey1=val1,key2=val2, …
```

Specify comma-separated key=value pairs to govern the behavior of the adapter. This includes a `configfilesection`=[`section`] key-value pair that specifies a section label in the

configuration file. For more information, see "Setting Configuration Parameters in a File (Page 210)".

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotation marks (for example, `key=\"str,ing\"`).

You can mix and match parameters using `configfilesection=[section]` and other `key=value` pairs. Key-value pairs override values specified in the configuration file.

For example, here, a file and socket adapter parses the section named **adaptercsvAdapterSubConfigfile_input** in connectors.config for the parameters to govern adapter behavior:

```
dfesp_fs_adapter -C
configfilesection=[adaptercsvAdapterSubConfigfile_input]
```

Here, all the parameters that govern the behavior of the file and socket adapter are specified as key-value pairs on the command line:

```
dfesp_fs_adapter -C
type=sub,host=localhost,port=49900,project=project_01,
continuousquery=cq_01,window=input,snapshot=true,
fsname=TEST_OUTPUT/output/input.out,fstype=csv,dateformat=”%Y-%m-
%d   %H:%M:%S”
```

Here, the file and socket adapter parses the section named **adaptercsvAdapterSubConfigfile_input** in connectors.config to govern adapter behavior. However, the key-value pairs specified on the command line override any values specified in that section.

```
dfesp_fs_adapter -C
configfileinput=[adaptercsvAdapterSubConfigfile_input],
fsname=TEST_OUTPUT/output/input.out,fstype=csv,dateformat="%Y-%m-%d
%H:%M:%S"
```

2.3.2 Using the Adapter Connector

The Adapter Connector functions as a wrapper around an adapter, enabling the Edge Streaming Server to process the adapter as a connector. Using the Adapter Connector, you can orchestrate adapters as you orchestrate connectors. This orchestration enables you to better deploy adapters in scale. (For more information about connector orchestration, see “Orchestrating Connectors (Page 14)”.)

You can also use the Adapter Connector to start and end adapter processes. This enables you to manage the life cycle of an adapter with the Edge Streaming Server, regardless of any need for orchestration.

The Adapter Connector sets its state to RUNNING while the associated adapter is running and to FINISHED when the adapter is not running. When the connector stops, the associated running adapter ends.

The following table describes the required and optional parameters in the XML project definition for the Adapter Connector:

Table 2-2 Required Parameters for the Adapter Connector

Parameter	Description
command	Specifies the command and options that are used to run the adapter from the command line. All required adapter parameters must be specified when defining the command parameter. Include the -k parameter for adapters that require it to specify the mode. Do not include -h in the command parameter definition here. It is overwritten by the URL specified by the url parameter or the generated default URL.
type	Specifies the mode of the adapter. The mode can be "pub" or "sub" and must match the mode of the running adapter.

Table 2-3 Optional Parameters for the Adapter Connector

Parameter	Description
url	Specifies the URL for the adapter connection. If this parameter is not included, a default URL is generated. Include the adapter parameter along with the URL for the window. For example, if you are using the Twitter adapter, you could write -u "myserver.company.com:9951/proj/cq/ src".

For example, the following Adapter Connector encapsulates a Cassandra adapter.

```
<connector class='adapter' name='cassandra_pub'>
<properties>
<property name='command'>@DFESP_HOME@/bin/dfesp_cassandra_publisher
-f @CASSANDRA_HOST@
-x @CASSANDRA_PORT@ -k Sesptest -t pubSimple -e "select * from
Sesptest.pubSimple"
</property>
<property  name='type'>pub</property>
<property  name='url'>-u  dfESP://esaserv1:59999/project_01/cq_01/
pub_win</property>
</properties>
```

```
</connector>
```

Note

When the Adapter Connector starts, the submitted command for the associated adapter appears in the log. To troubleshoot the Adapter Connector, look for the submitted command in the log and verify that all the parameters are correct.

IoT Protocols for Edge Streaming Server 3

3.1 MQTT Connector and Adapter

3.1.1 Using the MQTT Connector

MQ Telemetry Transport (MQTT) is a simple and lightweight publish/subscribe messaging protocol, designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. The design principle is to minimize network bandwidth and device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery. This principle makes the protocol suitable for the emerging "machine-to- machine" (M2M) or "Internet of Things" world of connected devices. It is also suitable for mobile applications, where bandwidth and battery power are at a premium.

Note

To use this connector:

- Locate the reference to this connector in the `connectors: excluded:` section of the file **esp- properties.yml**.
- Set the value to **false**.

The MQTT connector communicates with an MQTT broker for both publish and subscribe operations, and provides the following capabilities:

- Subscribe to an MQTT broker topic and publish received messages into a Source window.
- Subscribe to a window and publish received events to an MQTT broker topic.
- Auto-reconnect to the MQTT broker in case of lost connection.
- Transport Layer Security (TLS) v1.2 encryption and authentication for the MQTT server connections.
- Event as transmitted through the MQTT broker can be encoded as ESA binary event blocks, CSV, JSON, Google Protocol buffers, and opaque string message formats.
- Supports MQTT protocol v3.1.1 .

You must install the Eclipse Mosquitto Client library v1.4.5 run-time libraries on the platform that hosts the running instance of the connector. It provides the fastest interface to MQTT brokers. This library is available for all platforms and can be downloaded from http://mosquitto.org/download. The run-time environment must define the path to those libraries (for example, specifying **LD_LIBRARY_PATH**).

If required, you can configure the mqttpassword parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. When you install the Edge Streaming Analytics System Encryption and Authentication Overlay, you install the included OpenSSL executable. Use the following command on the console to invoke OpenSSL to display your encrypted password:

```
echo "mqttpassword" | openssl enc -e -aes-256-cbc -a -salt -pass
pass:"espMQTTconnectorUsedByUser=mqttuserid"
```

Then copy the encrypted password into your `mqttpassword` parameter and enable the `mqttpasswordencrypted` parameter.

Table 3-1 Required Parameters for Subscriber MQTT Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`snapshot`	Specifies whether to send snapshot data.When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.
`mqtthost`	Specifies the MQTT server host name.
`mqttclientid`	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, `mqttdonotcleansession` must be false. Must be unique among all clients connected to the MQTT server.
`mqtttopic`	Specifies the string to use as an MQTT topic to publish events to.
`mqttqos`	Specifies the requested Quality of Service. Values can be 0, 1 or 2.
`mqttmsgtype`	Specifies `binary`, `csv`, `json`, `protobuf`, or the name of a string field in the subscribed window schema.

Table 3-2 Required Parameters for Publisher MQTT Connectors

Parameter	Description
`type`	Specifies to publish. Must be "`pub`".
`mqtthost`	Specifies the MQTT server host name.
`mqttclientid`	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, `mqttdonotcleansession` must be false. Must be unique among all clients connected to the MQTT server.
`mqtttopic`	Specifies the string to use as an MQTT subscription topic pattern.
`mqttqos`	Specifies the requested Quality of Service. Values can be 0, 1 or 2.
`mqttmsgtype`	Specifies `binary, csv, json, protobuf,` or `opaquestring`. For opaquestring, if the Source window schema has 3 fields, it is assumed to be `index:int64,topic:string,message:string`. If the Source window schema has 2 fields, it is assumed to be `index:int64,message:string`.

Table 3-3 Optional Parameters for Publisher MQTT Connectors

Parameter	Description
mqttuserid	Specifies the user name required to authenticate the connector's session with the MQTT server.
mqttpassword	Specifies the password associated with mqttuserid.
mqttport	Specifies the MQTT server port. Default is 1883.
mqttacceptretainedmsg	Sets to true to accept to receive retained message. The default is "false".
mqttcleansession	Set to true to instruct the MQTT Server to clean all messages and subscriptions on disconnect, false to instruct it to keep them. The default is "true".
mqttkeepaliveinterval	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10.
publishwithupsert	Builds events with opcode=Upsert instead of Insert.
transactional	When mqttmsgtype=CSV, sets the event block type to transactional. The default event block type is normal.
blocksize	When mqttmsgtype=CSV, specifies the number of events to include in a published event block. The default value is 1.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
noautogenfield	Specifies that input events are missing the key field that is automatically generated by the Source window.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
protofile	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.This parameter is ignored when mqttmsgtype is not set to protobuf.
protomsg	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.This parameter is ignored when mqttmsgtype is not set to protobuf.
mqttssl	Specifies to use TLS to connect to the MQTT broker. The default is "false". In order to use TLS, the Edge Streaming Analytics encryption overlay must be installed.
mqttsslcafile	If mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified.

Parameter	Description
mqttsslcapath	If mqttssl=true, specifies the path to a directory containing the PEM encoded trusted CA certificate files. See mosquitto.conf for more details about configuring this directory. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcertfile	If mqttssl=true, specifies the path to a file containing the PEM encoded certificate file for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslkeyfile	If mqttssl=true, specifies the path to a file containing the PEM encoded private key for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslpassword	If mqttssl=true, and if key file is encrypted, specifies the password for decryption.
configfilesection	Specifies the name of the section in /opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
mqttpasswordencrypted	Specifies that mqttpassword is encrypted.
maxevents	Specifies the maximum number of events to publish.
stripnullsfromcsv	Strip all nulls from input CSV events.

Table 3-4 Optional Parameters for Subscriber MQTT Connectors

Parameter	Description
mqttuserid	Specifies the user name required to authenticate the connector's session with the MQTT server.
mqttpassword	Specifies the password associated with mqttuserid.
mqttport	Specifies the MQTT server port. The default is 1883.
mqttretainmsg	Sets to true to make the published message retained in the MQTT Server. The default is "false".
mqttdonotcleansession	Instructs the MQTT Server to keep all messages and subscriptions on disconnect, instead of keeping them. The default is "false".
mqttkeepaliveinterval	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10.
mqttmsgmaxdeliveryattempts	Specifies the number of times the connector tries to resend the message in case of failure. The default is 20.
mqttmsgdelaydeliveryattempts	Specifies the delay in milliseconds between delivery attempts specified with mqttmsgmaxdeliveryattempts . The default is 500.
mqttmsgwaitbeforeretry	Specifies the number of seconds to wait before retrying to send messages to the MQTT broker. This applies to publish messages with QoS > 0. The default is 20.
mqttmaxinflightmsg	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
collapse	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.

Parameter	Description
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
protofile	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.This parameter is ignored when mqttmsgtype is not set to protobuf.
protomsg	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.This parameter is ignored when mqttmsgtype is not set to protobuf.
mqttssl	Specifies to use TLS to connect to the MQTT broker. The default is "false". In order to use TLS, the Edge Streaming Analytics encryption overlay must be installed.
mqttsslcafile	If mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcapath	If mqttssl=true, specifies the path to a directory containing the PEM encoded trusted CA certificate files. See mosquitto.conf for more details about configuring this directory. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcapath	If mqttssl=true, specifies the path to a directory containing the PEM encoded trusted CA certificate files. See mosquitto.conf for more details about configuring this directory. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcertfile	If mqttssl=true, specifies the path to a file containing the PEM encoded certificate file for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslkeyfile	If mqttssl=true, specifies the path to a file containing the PEM encoded private key for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslpassword	If mqttssl=true, and if key file is encrypted, specifies the password for decryption.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
mqttpasswordencrypted	Specifies that mqttpassword is encrypted.
maxevents	Specifies the maximum number of events to publish.
stripnullsfromcsv	Strip all nulls from input CSV events.

Table 3-5 Optional Parameters for Subscriber MQTT Connectors

Parameter	Description
`mqttuserid`	Specifies the user name required to authenticate the connector's session with the MQTT server.
`mqttpassword`	Specifies the password associated with mqttuserid.
`mqttport`	Specifies the MQTT server port. The default is 1883.
`mqttretainmsg`	Sets to true to make the published message retained in the MQTT Server. The default is "false".
`mqttdonotcleansession`	Instructs the MQTT Server to keep all messages and subscriptions on disconnect, instead of keeping them. The default is "false".
`mqttkeepaliveinterval`	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10.
`mqttmsgmaxdeliveryattempts`	Specifies the number of times the connector tries to resend the message in case of failure. The default is 20.
`mqttmsgdelaydeliveryattempts`	Specifies the delay in milliseconds between delivery attempts specified with mqttmsgmaxdeliveryattempts . The default is 500.
`mqttmsgwaitbeforeretry`	Specifies the number of seconds to wait before retrying to send messages to the MQTT broker. This applies to publish messages with QoS > 0. The default is 20.
`mqttmaxinflightmsg`	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile`	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.This parameter is ignored when mqttmsgtype is not set to protobuf.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.This parameter is ignored when mqttmsgtype is not set to protobuf.
`mqttssl`	Specifies to use TLS to connect to the MQTT broker. The default is "false". In order to use TLS, the Edge Streaming Analytics encryption overlay must be installed.

Parameter	Description
mqttsslcafile	If mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcapath	If mqttssl=true, specifies the path to a directory containing the PEM encoded trusted CA certificate files. See mosquitto.conf for more details about configuring this directory. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcertfile	If mqttssl=true, specifies the path to a file containing the PEM encoded certificate file for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslkeyfile	If mqttssl=true, specifies the path to a file containing the PEM encoded private key for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslpassword	If mqttssl=true, and if key file is encrypted, specifies the password for decryption.
csvincludeschema	Specifies never, once, or pereventblock. The default value is "never". When mqttmsgtype = CSV, prepend output CSV with the window's serialized schema.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
mqttpasswordencrypted	Specifies that mqttpassword is encrypted.
addcsvopcode	Prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

3.1.2 Using the MQTT Adapter

Overview

The MQTT adapter supports publish and subscribe operations against an MQTT server. You must install the Eclipse Mosquitto Client libraries to use the adapter.

dfesp_mqtt_adapter -C *key1*=*val1*,*key2*=*val2*,*key3*=*val3*,…

Note

- If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").
- The MQTT adapter is not supported on Windows systems.

Subscriber Usage

Table 3-6 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form dfESP://host:port/project/continuousquery/window.Append ?snapshot=true | false for subscribers.Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`mqtthost=hostname`	Specifies the MQTT server host name.
`mqtttopic=topic`	Specifies the MQTT topic. An MQTT topic is an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).
`mqttclientid=string`	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, mqttdonotcleansession must be false. Must be unique among all clients connected to the MQTT server.
`mqttqos=0 | 1 | 2`	Specifies the requested Quality of Service.
`mqttmsgtype=binary | csv | json | protobuf | opaquestringfield`	Specifies the message format or the name of a string field in the subscribed window schema.

Table 3-7 Optional Keys

Key-Value Pair	Description
`mqttuserid=username`	Specifies the user name required to authenticate the session with the MQTT server.
`mqttpassword=password`	Specifies the password associated with mqttuserid.
`mqttport=port`	Specifies the MQTT server port. The default is 1883.
`mqttdonotcleansession=true | false`	When true, instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is "false".
`mqttkeepaliveinterval=seconds`	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10 seconds.

Key-Value Pair	Description
`mqttssl=true | false`	When true, uses Transport Layer Security (TLS) to connect to the MQTT broker. The default is "false". In order to use TLS, the Edge Streaming Analytics System Encryption overlay encryption overlay must be installed.
`mqttsslcafile=path`	When mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified.
`mqttsslcapath=path`	When mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified. See mosquitto.conf for more information about configuring this directory.
`mqttsslcertfile=path`	When mqttssl=true, specifies the path to a file containing the encoded certificate file for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
`mqttsslkeyfile=path`	When mqttssl=true, specifies the path to a file containing the encoded private key for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
`mqttsslpassword=password`	When mqttssl=true and mqttsslkeyfile is encrypted, specifies the password for decryption.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the .proto file to be used for Google protocol buffer support.This key is ignored when mqttmsgtype is not set to protobuf.
`protomsg=message`	Specifies the message itself in the .proto file that is specified by the protofile parameter.This key is ignored when mqttmsgtype is not set to protobuf.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`mqttpasswordencrypted=true | false`	When true, mqttpassword is encrypted

Key-Value Pair	Description
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. Note: No transport configuration file is required for native transport.
`mqttretainmsg=true | false`	When true, make the published message retained in the MQTT Server. The default is "false".
`mqttmsgmaxdeliveryattempts=number`	Specifies the number of times to try to resend the message in case of failure. The default is 20.
`mqttmsgdelaydeliveryattempts=mil liseconds`	Specifies the delay in milliseconds between delivery attempts specified in mqttmsgmaxdeliveryattempts. The default is 500.
`mqttmsgwaitbeforeretry=seconds`	Specifies the number of seconds to wait before retrying to send messages to the MQTT broker (applies only to messages with QoS > 0). The default is 20.
`mqttmaxinflightmsg=number`	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
`csvincludeschema=never | once | pereventblock`	When mqttmsgtype=CSV, specifies when to pass window schema to CSV subscriber callback. The default value is "never".
`csvmsgperevent=true | false`	When true, send one message per event. By default, one message is sent per transactional event.
`csvmsgpereventblock=true | false`	When true, send one message per event block. By default, one message is sent per transactional event block.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Publisher Usage

Table 3-8 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form dfESP://host:port/project/continuousquery/window.
`mqtthost= hostname`	Specifies the MQTT server host name.
`mqtttopic=topic`	Specifies the MQTT topic. An MQTT topic is an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).
`mqttclientid=string`	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, mqttdonotcleansession must be false. Must be unique among all clients connected to the MQTT server.

Key-Value Pair	Description
`mqttqos=0 | 1 | 2`	Specifies the requested Quality of Service.
`mqttmsgtype=binary | csv | json | protobuf | opaquestringfield`	Specifies the message format. When the Source window schema has three fields, the value of opaquestringfield is assumed to be "index:int64,topic:string,message:string". When the Source window schema has two fields, the value of opaquestringfield is assumed to be "index:int64,message:string".

Table 3-9 Optional Keys

Key-Value Pair	Description
`mqttuserid=username`	Specifies the user name required to authenticate the session with the MQTT server.
`mqttpassword=password`	Specifies the password associated with mqttuserid.
`mqttport=port`	Specifies the MQTT server port. The default is 1883.
`mqttdonotcleansession=true | false`	When true, instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is "false".
`mqttkeepaliveinterval=seconds`	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10 seconds.
`mqttssl=true | false`	When true, uses TLS to connect to the MQTT broker. The default is "false". In order to use TLS, the Edge Streaming Analytics encryption overlay must be installed.
`mqttsslcafile=path`	When mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified.
`mqttsslcapath=path`	When mqttssl=true, specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified. See mosquitto.conf for more information about configuring this directory.
`mqttsslcertfile=path`	When mqttssl=true, specifies the path to a file containing the encoded certificate file for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
`mqttsslkeyfile=path`	When mqttssl=true, specifies the path to a file containing the encoded private key for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
`mqttsslpassword=password`	When mqttssl=true and mqttsslkeyfile is encrypted, specifies the password for decryption.
`dateformat=format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields `are` interpreted as an integer number of seconds (DATE) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=file`	Specifies the .proto file to be used for Google protocol buffer support.This key is ignored when mqttmsgtype is not set to protobuf.

Key-Value Pair	Description
`protomsg=message`	Specifies the message itself in the .proto file that is specified by the protofile parameter.This key is ignored when mqttmsgtype is not set to protobuf.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [`configfilesection`].
`mqttpasswordencrypted=true | false`	When true, mqttpassword is encrypted.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. Note: No transport configuration file is required for native transport.
`mqttacceptretainedmsg= true | false`	When true, enables receiving of retained messages. The default is "false".
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=delimeter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, specifies that input events are missing the key field that is autogenerated by the Source window.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`addcsvopcode= true | false`	When true, prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is true.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).

Key-Value Pair	Description
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

3.2 OPC UA Connector and Adapter

3.2.1 Using the OPC UA Connector

Overview

OPC Unified Architecture (OPC UA) is a machine-to-machine communication protocol for industrial automation. It is used for communication with industrial equipment and systems for data collection and control. Industries that use OPC UA include pharmaceutical, oil and gas, building automation, industrial robotics, security, manufacturing, and process control.

Using the OPC-UA Connector

The OPC-UA connector communicates with an OPC-UA server for publish and subscribe operations against the Value attribute of a fixed set of OPC-UA nodes. The OPC-UA server endpoint is a required connector configuration parameter.

The set of OPC-UA nodes must belong to a common namespace in the OPC-UA server. By default, the connector uses the namespace with namespace index=0. You can configure an optional connector parameter to specify a different namespace URI. To access additional nodes in a different server namespace, you must run a separate instance of the connector attached to a different Source window.

The set of OPC-UA nodes accessed by the connector is defined by the schema of the window attached to the connector. By default, the window schema field names must specify the Node ID of an OPC-UA node in the configured OPC-UA server namespace. The appropriate Node IDs to configure on the connector can be found using a standard OPC-UA browsing tool against the target OPC-UA server.

Alternatively, you can specify an optional connector parameter that maps each window field name to a Node ID. In that case, the window fields can be named as desired.

The Node ID format is a fixed notation that contains the node identifier type and the node identifier, in the form of s_MyTemperature. The first character is the identifier type and must be

's' (String), 'i' (Integer), 'g' (GUID), or 'b' (Opaque). Following the underscore is the identifier string. If the type is Opaque, then the string must be base64–encoded. The window schema field types must match the type of the corresponding OPC-UA node.

The supported mappings are:

Table 3-10 Supported Mappings for a Publisher:

ESA type	OPC-UA type
INT32	INT32 or UINT32 or INT16 or UINT16 or BOOLEAN
INT64	INT64 or UINT64 or INT32 or UINT32 or INT16 or UINT16 or BOOLEAN
DOUBLE	DOUBLE or FLOAT or BOOLEAN
STRING	STRING
DATE	DATETIME
STAMP	DATETIME
MONEY	Not supported

Table 3-11 Supported Mappings for a Subscriber:

ESA type	OPC-UA type
INT32	INT32 or UINT32
INT64	INT64 or UINT64
DOUBLE	DOUBLE
STRING	STRING
DATE	DATETIME
STAMP	DATETIME
MONEY	Not supported

The OPC-UA publisher connector requires one additional field at the front of the Source window schema. This additional field is a 64-bit integer that is incremented by the connector with every published event. You can use this field as the window key field if no other field is appropriate.

The publisher publishes an event when the value of the Value attribute of an OPC-UA node in the Source window schema changes. If the Source window contains additional fields representing other OPC-UA nodes, the values in those fields remain unchanged. By default, the event contains an Insert opcode unless you configure the connector to use Upsert instead.

You can also configure an optional publisher parameter that specifies an interval in seconds. Each expiration of this interval generates a fetch of the current value of all OPC-UA nodes in the Source window. An event containing those values is published.

The OPC-UA subscriber connector writes the values in all fields of a subscribed event to the Value attribute of the corresponding OPC-UA node when the event is generated by the subscribed window. It ignores all opcodes except Insert and Update.

Table 3-12 Required Parameters for OPC-UA Connectors

Parameter	Description
`type`	Specifies to publish or subscribe.
`opcuaendpoint`	Specifies the OPC-UA server endpoint (only the portion following opc.tcp://).
`snapshot`	Specifies whether to send snapshot data.When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 3-13 Optional Parameters for Publisher OPC-UA Connectors

Parameter	Description
`opcuanamespaceuri`	Specifies the OPC-UA server namespace URI. The default is the namespace at index=0.
`opcuausername`	Specifies the OPC-UA user name. By default, there is none.
`opcuapassword`	Specifies the OPC-UA password. By default, there is none.
`opcuanodeids`	Specifies a comma-separated list of Node IDs to map to ESA window schema fields, in the form <identifier type>_<identifier>. The list size must be equal to the number of fields in the subscribed window schema - 1. Window field names are in Node ID form by default.
`publishinterval`	Specifies an interval in seconds when current values of all nodes in the Source window schema are published. The default is to publish when one or more values changes.
`transactional`	Sets the event block type to transactional. The default value is "normal".
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/ config/ etc/ EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`publishwithupsert`	Builds events with opcode=Upsert instead of Insert.
`maxevents`	Specifies the maximum number of events to publish.
`opcuanodeiddelimiter`	Specifies the character used to delimit fields in theopcuanodeids parameter. The default value is ','.

Table 3-14 Optional Parameters for Subscriber OPC-UA Connectors

Parameters	Description
`opcuanamespaceuri`	Specifies the OPC-UA server namespace URI. The default is the namespace at index=0.
`opcuausername`	Specifies the OPC-UA user name. By default, there is none.
`opcuapassword`	Specifies the OPC-UA password. By default, there is none.
`opcuanodeids`	Specifies a comma-separated list of Node IDs to map to ESA window schema fields, in the form <identifier type>_<identifier>. The list size must be equal to the number of fields in the subscribed window schema. Window field names are in Node ID form by default.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/ config/ etc/ EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`opcuanodeiddelimiter`	Specifies the character used to delimit fields in theopcuanodeids parameter. The default value is ','.

3.2.2 Using the OPC UA Adapter

Overview

The OPC-UA publisher and subscriber adapter and the OPC-UA publisher and subscriber connector share configuration parameters, with the exception of some adapter-only optional parameters.

```
dfesp_opcua_adapter -C key1=val1,key2=val2,key3=val3,…
```

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Subscriber Usage

Table 3-15 Required Key

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false for subscribers.
`opcuaendpoint=endpoint`	Specifies the OPC-UA server endpoint, without the opc.tcp:// prefix.

Table 3-16 Optional Keys

Key-Value Pair	Description
`opcuanamespaceuri=uri`	Specifies the OPC-UA server namespace URI. The default is the namespace at index=0.
`opcuausername=username`	Specifies OPC-UA user name. By default, there is none.
`opcuapassword=password`	Specifies the OPC-UA password. By default, there is none.
`opcuanodeids=list`	Specifies a comma–separated list of Node IDs to map to ESA window schema fields, in the form <identifier type>_<identifier>. The default assumes that window field names are in Node ID form.
`gdconfig=list`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [`configfilesection`].
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.

Key-Value Pair	Description
transportconfigfile=file	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. Note: No transport configuration file is required for native transport.
opcuanodeiddelimiter=character	Specifies the character used to delimit fields in the opcuanodeidsparameter. The default value is ','.

Publisher Usage

Table 3-17 Required Keys

Key-Value Pair	Description
type=pub	Specifies a publisher adapter
url=pubsubURL	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window.**
opcuaendpoint=endpoint	Specifies the OPC-UA server endpoint, without the opc.tcp://prefix.
opcuanamespaceuri=uri	Specifies the OPC-UA server namespace URI. The default is the namespace at index=0.
opcuausername=username	Specifies OPC-UA user name. By default, there is none.
opcuapassword=password	Specifies the OPC-UA password. By default, there is none.
opcuanodeids=list	Specifies a comma–separated list of Node IDs to map to ESA window schema fields, in the form **<**identifier type**>_<**identifier**>**. The default assumes that window field names are in Node ID form.
gdconfig=file	Specifies the guaranteed delivery configuration file.
transport=native | solace | tervela | rabbitmq | kafka	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
loglevel=trace | debug | info | warn | error | fatal | off	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
logconfigfile=file	Specifies the log configuration file.
tokenlocation=location	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
configfilesection=[section]	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/EdgeStreamingServer/ default/connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Key-Value Pair	Description
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg.** For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. Note: No transport configuration file is required for native transport.
`opcuanodeiddelimiter=character`	Specifies the character used to delimit fields in the opcuanodeidsparameter. The default value is ','.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When `true`, events are transactional.
`publishwithupsert=true | false`	When `true`, build events with opcode = Upsert instead of Insert.
`publishinterval=seconds`	Specifies the interval (in seconds) at which all current Node ID values are published into the Source window. By default, values are published when they change.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true`, quiesces the project after all events are injected into the Source window.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

3.3 Modbus Connector and Adapter

3.3.1 Using the Modbus Connector

Overview

Modbus is a serial communications protocol commonly used to connect industrial electronic devices. Modbus enables communication among devices connected to the same network. For example, you can use Modbus in a system to measure temperature and humidity and then communicate results to a central server.

Using the Modbus Connector

The Modbus connector supports publish and subscribe operations to and from Edge Streaming Analytics through the Modbus protocol.

The Modbus connector uses four object types. Each object type has a value for addresses between 0 and 65,534.

Table 3-18 Modbus Connector Object Types

Object Type	Read or Write	Value
Coil	Read and Write	8 bit (on or off)
Input	Read Only	8 bit (on or off)
Input Register	Read Only	16 bit
Holding Register	Read and Write	16 bit

The Modbus publisher connector reads values for each of the object types from the Modbus server and publishes the values to Edge Streaming Analytics. The Modbus subscriber connector reads data from Edge Streaming Analytics and publishes the values to Modbus. The subscriber supports only the Read-Only object types coil and holding register.

The Modbus connector uses the libmodbus library to communicate with the Modbus server. For more information about libmodbus, see http://libmodbus.org/ (http://libmodbus.org/).

You can use the Modbus PLC Simulator to simulate a Modbus instance. The Modbus PLC simulator is a Windows executable that supports the Modbus protocol. With the simulator, you can view and modify the current values for all addresses for Modbus objects. The Modbus connector communicates with the simulator to read and write Modbus addresses through the libmodbus library. You can download the simulator at http:// www.plcsimulator.org/ (http:// www.plcsimulator.org/).

The Modbus publisher connector polls the Modbus server for data and publishes the values to a Source window on a running Edge Streaming Server. To receive data from a Modbus connector, the Source window must have the following schema:

```
type*:string,address*:int32,value:int32
```

- `type` is the Modbus object type (coil, input, input register, or holding register)
- `address` is the Modbus object address (between 0 and 65,534)
- `value` is the Modbus object value (0 or 1 for coil and input, 16–bit value for registers)

If the publisher connector reads a non-0 value, it publishes an Upsert event into the model, such as p,n,holding-register,1,118. If the publisher connector reads a 0 value, it publishes a Delete event into the model, such as d,n,holding-register,1,0.

The Modbus subscriber connector subscribes to a window on a running Edge Streaming Server and publishes data to the Modbus server. The window subscribed to must also include the type, address, and value fields in its schema.

When the subscriber connector receives an event from Edge Streaming Analytics, the values are pulled from the event and sent to Modbus using the libmodbus library. If you are running the simulator, the values update immediately.

You can choose to specify the data types and addresses to poll. When specifying multiple addresses, separate each address with a comma. The items in the list are either individual addresses or address ranges. For example, to poll holding registers for addresses 10 through 20, 33, and 44, define the connector property as follows:

```
<property name='holdingRegisters'>10-20,33,44</property>
```

Note

If you specify a data type but no addresses, the connector scans through all available addresses.

Table 3-19 Required Parameters for the Modbus Publisher Connector

Parameter	Description
`type`	Specifies to publish. Must be `pub`.
`modbus`	Specifies the Modbus server host. Instead of a host name, you can specify `host:port`. If no port is specified, the default is 502.

Table 3-20 Optional Parameters for the Modbus Publisher Connector

Parameter	Description
`interval`	Specifies the interval, in seconds, at which the object value data is pulled from the Modbus server. The default is 10.
`coils`	Specifies the coil addresses to poll.
`inputs`	Specifies the input addresses to poll.
`inputRegisters`	Specifies the input register addresses to poll.
`holdingRegisters`	Specifies the holding register addresses to poll.
`slaveId`	Specifies the device slave ID in the Modbus environment.

Table 3-21 Required Parameters for the Modbus Subscriber Connector

Parameter	Description
`type`	Specifies to subscribe. Must be `sub`.
`modbus`	Specifies the Modbus server host. Instead of a host name, you can specify a port with the form host:port. If no port is specified, the default is 502.
`snapshot`	If true, pulls a snapshot from the window at start-up.

Table 3-22 Optional Parameters for the Modbus Subscriber Connector

Parameter	Description
`slaveId`	Specifies the device slave ID in the Modbus environment.

3.3.2 Using the Modbus Adapter

Overview

The Modbus adapter supports publish and subscribe operations between Edge Streaming Analytics and a Modbus server.

In publisher mode, the adapter reads data from Modbus and publishes it to a Source window on a running Edge Streaming Server. A Source window receiving data from a Modbus publisher adapter must follow the same schema rules as a Source window receiving events through a Modbus publisher connector.

dfesp_modbus_adapter -C *key1=val1,key2=val2,key3=val3,…*

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Subscriber Usage

In subscriber mode, the adapter receives events from a window on a running Edge Streaming Server and publishes the data to Modbus. A window sending data to a Modbus subscriber adapter must follow the same schema rules as a window sending events through a Modbus subscriber connector.

Table 3-23 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`modbus=host`	Specifies the Modbus server host name. The default port is 502. To specify a different port, use the form host:port.
`slaveId=id`	Specifies the slave ID of the Modbus device.

Table 3-24 Optional Keys

Key-Value Pair	Description
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported.
`gdconfig=gdconfigfile`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Publisher Usage

Table 3-25 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**.
`modbus=host`	Specifies the Modbus server host name. The default port is 502. To specify a different port, use the form host:port.
`slaveId=id`	Specifies the slave ID of the Modbus device.

Table 3-26 Optional Keys

Key-Value Pairs	Description
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported.
`gdconfig=gdconfigfile`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Key-Value Pairs	Description
tokenlocation=location	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
configfilesection=[section]	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
blocksize=size	Specifies the event block size. The default value is 1.
maxevents=maxevents	Specifies the maximum number of events to publish.
interval=seconds	Specifies the Modbus polling interval, in seconds. The default is 10.
coils=coils	Specifies the Modbus coils to read.
inputs=inputs	Specifies the Modbus inputs to read.
inputRegisters=registers	Specifies the Modbus input registers to read.
holdingRegisters=hregisters	Specifies the Modbus holding registers to read.

3.4 Bacnet Connector and Adapter

3.4.1 Using the Bacnet Connector

Overview

BACnet is a communications protocol for Building Automation and Control (BAC) networks based on the ISO 16484-5 standard protocol. You can use BACnet to enable smart applications for HVAC, lighting control, access control, and fire detection systems.

Using the BACnet Connector

The BACnet connector polls BACnet devices for data from a predefined set of BACnet objects for publish operations into an stream processing Source window.

Note

Support for the BACnet connector is available only on Linux platforms.

The set of BACnet devices and their objects is defined in a local JSON configuration file. The path to the JSON configuration file is specified with the required connector configuration parameter bacnetconfigfile. The polling interval for each object is defined by the JSON Period key value in the configuration file. The format for the JSON configuration file is as follows:

```
[
{
```

```
"RemoteEndPoint": "<device ipaddr:port>",
"MaxAPDULengthAccepted": <max apdu length>,
"LocalPort": <local port to use with this device>,
"NetworkNumber": <bacnet network number (optional)>,
"MAC": "<bacnet mac address (optional)>",
"BACnetPoints": [
{
"Topic": "<any string unique to this point>",
"ObjectIdentifier": <bacnet object identifier>,
"PropertyIdentifier": <bacnet property identifier>,
"Period": <polling interval in seconds>
},
…
]
},
…
]
```

To register the connector as a foreign device and directly access BACnet devices, configure a BACnet-IP Broadcast Management Device (BBMD) IP address and port.

Multiple BACnet connector instances cannot exist within a single process. To run multiple BACnet connector instances as individual processes, you must run multiple BACnet adapters. You can run these adapters manually as separate processes or as multiple Adapter Connectors (Page 20) configured in the edge streaming analytics model.

Table 3-27 Required Fields in the Source Window Schema of the BACnet Connector:

Schema Field	Description
`id:int64`	Key field, controlled by the connector
`adapterid:string`	Key field, controlled by the connector
`topic:string`	Copied from JSON Topic key value
`timestamp:stamp`	Timestamp indicating when the event was built

Define any number of additional fields to contain values read from BACnet objects. When a BACnet object is read according to its polling interval, the received value is copied into the user-defined fields compatible with the value type, and a corresponding streaming event is built. If there is no matching field for a received value type, a fatal error is reported.

Table 3-28 Required Parameters for the Publisher BACnet Connector

Parameter	Description
`bacnetbbmdaddress`	Specifies the IP address of the BBMD
`bacnetbbmdport`	Specifies the port of the BBMD
`bacnetconfigfile`	Specifies the JSON configuration file containing BACnet device and object

Table 3-29 Optional Parameters for the Publisher BACnet Connector

Parameter	Description
`bacnetipport`	Specifies the local port used by the connector. The default port number is 47808.
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`ignoretimeouts`	Logs a warning and continues if an attempt to read a property from a BACnet device results in a time-out. The default is to log an error and stop.
`publishwithupsert`	Builds events with opcode=Upsert instead of Insert.
`maxevents`	Specifies the maximum number of events to publish.
`transactional`	Sets the event block type to transactional. The default event block type is normal.

Table 3-30 BACnet to Edge Streaming Analytics Data Type Mappings

BACnet type	Edge Streaming Analytics data types
`Boolean`	ESP_INT32 ESP_INT64 ESP_DOUBLE
`Unsigned Int`	ESP_INT32 ESP_INT64 ESP_DOUBLE
`Signed Int`	ESP_INT32 ESP_INT64 ESP_DOUBLE
`Real`	ESP_DOUBLE
`Double`	ESP_DOUBLE
`Character String`	ESP_UTF8STR
`Date`	ESP_DATETIME ESP_TIMESTAMP
`Enumerated`	ESP_INT32 ESP_INT64 ESP_DOUBLE
`Octet String`	ESP_BINARY

3.4.2 Using the Bacnet Publisher Adapter

The BACnet publisher adapter provides the same functionality as the BACnet connector (Page 46), in addition to several adapter-only optional parameters.

dfesp_bacnet_adapter -C *key1*=*val1*,*key2*=*val2*,*key3*=*val3*,…

Note

- Support for the BACnet adapter is available only on Linux platforms.
- If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 3-31 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`bacnetbbmdaddress=BBMDipaddr`	Specifies the IP address of the BBMD.
`bacnetbbmdport=BBMDport`	Specifies the local port used by the adapter. The default port number is 47808.
`bacnetconfigfile=jsonfile`	Specifies the JSON configuration file containing BACnet device and object information.

Table 3-32 Optional Keys

Key-Value Pair	Description
`gdconfig=gdconfigfile`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files. These files are specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=logconfigfile`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for connection parameters.
`blocksize=evntblcksize`	Sets the event block size. The default value is 1.
`transactional=true | false`	When true, specifies that events are transactional.
`publishwithupsert=true | false`	When true, builds events with opcode=Upsert instead of Insert.

Key-Value Pair	Description
`restartonerror=true | false`	When true, restarts the adapter after a fatal error is reported.
`bacnetipport=localport`	Specifies the local port used by the adapter. The default port number is 47808.
`transportconfigfile=file`	Specifies the transport configuration file. The default value depends on the transport type specified. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. Note: No transport configuration file is required for native transport.
`ignoretimeouts=true | false`	When true, logs a warning and continues if an attempt to read a property from a BACnet device results in a time-out. The default is to log an error and stop.
`maxevents=maxevents`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When maxevents is configured, quiesces the project after all events are injected into the Source window.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

Messaging protocols for Edge Streaming Server 4

4.1 Rabbit MQ Connector and Adapter

4.1.1 Using the Rabbit MQ Connector

Overview

RabbitMQ is a lightweight open-source message broker. You can deploy it on premises and in the cloud. It supports multiple messaging protocols.

Using the Rabbit MQ Connector

The Rabbit MQ connector communicates with a Rabbit MQ server for publish and subscribe operations. The bus connectivity provided by the connector eliminates the need for the engine to manage individual publish/ subscribe connections. The connector achieves a high capacity of concurrent publish/subscribe connections to a single engine.

Note

To use this connector:

- Locate the reference to this connector in the connectors: excluded: section of the file **esp-properties.yml**.
- Set the value to false.

A Rabbit MQ subscriber connector receives event blocks and publishes them to a Rabbit MQ routing key. A Rabbit MQ publisher connector reads event blocks from a dynamically created Rabbit MQ queue and injects them into an edge stream processing Source window.

Event blocks as transmitted through the Rabbit MQ server can be encoded as binary, CSV, Google protobufs, or JSON messages. The connector performs any conversion to and from binary format. The message format is a connector configuration parameter.

The Rabbit MQ connector supports hot failover operation. This mode requires that you install the presence- exchange plug-in on the Rabbit MQ server. You can download that plug-in from https://github.com/tonyg/ presence-exchange.

Ensure that the presence-exchange version that you use matches that of the installed Rabbit MQ server. For example, if you use server version 3.5.x, you can use presence-exchange version 3.5.y, where x and y differ but both are version 3.5. Mismatched versions (for example, 3.4.x and 3.3.y) might work in some cases, but this type of mismatch is not recommended.

A corresponding edge stream processing publish/subscribe client plug-in library is available. This library enables a standard edge stream processing publish/subscribe client application to exchange event blocks with an edge stream processing server through a Rabbit MQ server.

The exchange takes place through the Rabbit MQ server instead of through direct TCP connections. To enable this exchange, add a call to C_dfESPpubsubSetPubsubLib().

When configured for hot failover operation, the active/standby status of the connector is coordinated with the Rabbit MQ server. Thus, a standby connector becomes active when the active connector fails. All involved connectors must meet the following conditions to guarantee successful switchovers:

- They must belong to identical Edge Streaming Analytics models.
- They must initiate message flow at the same time. This is required because message IDs must be synchronized across all connectors.

When a new subscriber connector becomes active, outbound message flow remains synchronized. This is due to buffering of messages by standby connectors and coordination of the resumed flow with the Rabbit MQ server. The size of the message buffer is a required parameter for subscriber connectors.

You can configure a subscriber Rabbit MQ connector to send a custom snapshot of window contents to any subscriber client. The client must have established a new connection to the Rabbit MQ server. This enables late subscribers to catch up upon connecting. This functionality also requires that you install the presence-exchange plug-in on the Rabbit MQ server.

When the connector starts, it subscribes to topic urlhostport/M (where urlhostport is a connector configuration parameter). This enables the connector to receive metadata requests from clients that publish or subscribe to a window in an engine associated with that host:port combination. Metadata responses consist of some combination of the following:

- project name of the window associated with the connector
- query name of the window associated with the connector
- window name of the window associated with the connector
- the serialized schema of the window

You must install Rabbit MQ client run-time libraries on the platform that hosts the running instance of the connector. The connector uses the rabbitmq-c v0.9.0 C libraries, which you can download from https://github.com/alanxz/rabbitmq-c. The run-time environment must define the path to those libraries (for example, specifying LD_LIBRARY_PATH on Linux platforms). If you build your rabbitmq-c libraries with Transport Layer Security (TLS) support enabled, make sure to include the path to those TLS libraries in your run-time environment.

For queues that are created by a publisher, the optional buspersistence parameter controls both auto-delete and durable.

Note

RabbitMQ server version 3.7.5 changed the default value for channel_max to 2047. This causes connection failures by the rabbitmq-c client because it tries to negotiate a value of 0. The workaround is to change the default channel_max value to 0 in your RabbitMQ server configuration.

Setting of the `buspersistence` parameter	Effect
true	auto-delete = false durable = true
false	auto-delete = true durable = false

The following holds when consuming from those queues:

Setting of the `buspersistence` parameter	Effect
true	exclusive = true noack = false
false	exclusive = false noack = true

When the publisher connector creates a durable receive queue with auto-delete disabled, it consumes from that queue with noack = false but does not explicitly acknowledge messages. This enables a rebooted edge stream processing server to receive persisted messages. To have a buspersistent publisher occasionally acknowledge groups of messages older than a specified age, configure the combination of ackwindow and acktimer parameters. This keeps the message queue from growing unbounded, avoiding administrator intervention.

The queue name is equal to the buspersistencequeue parameter appended with the configured topic parameter. The buspersistencequeue parameter must be unique on all publisher connectors that use the same Rabbit MQ exchange. The publisher connector enforces this by consuming the queue in exclusive mode when buspersistence is enabled.

For a subscriber connector, enabling buspersistence means that messages are sent with the delivery mode set to **persistent**.

For exchanges that are created by a publish or a subscribe, buspersistence controls only durable. That is, when buspersistence = true, durable = true, and when buspersistence = false, durable = false.

If required, you can configure the rmqpassword parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. If you have installed the Edge Streaming Analytics System Encryption and Authentication Overlay, you can use the included OpenSSL executable. Use the following command on the console to invoke OpenSSL to display your encrypted password:

```
echo "rmqpassword" | openssl enc -e -aes-256-cbc -a -salt -pass
pass:"espRMQconnectorUsedByUser=rmquserid"
```

Then copy the encrypted password into your rmqpassword parameter and enable thermqpasswordencrypted parameter.

Table 4-1 Required Parameters for Subscriber Rabbit MQ Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`rmquserid`	Specifies the user name required to authenticate the connector's session with the Rabbit MQ server.
`rmqpassword`	Specifies the password associated with rmquserid.
`rmqhost`	Specifies the Rabbit MQ server host name.
`rmqport`	Specifies the Rabbit MQ server port.
`rmqexchange`	Specifies the Rabbit MQ exchange created by the connector, if nonexistent.

Parameter	Description
`rmqtopic`	Specifies the Rabbit MQ routing key to which messages are published.
`rmqtype`	Specifies binary, csv, json, protobuf, or the name of a string field in the subscribed window schema.
`urlhostport`	Specifies the *host:port* field in the metadata topic subscribed to on start-up to field metadata requests.
`numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector. When exceeded, the oldest message is discarded. When the connector goes active, the buffer is flushed and buffered messages are sent to the fabric as required to maintain message ID sequence.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. This parameter is invalid if `buspersistence` or `hotfailover` is enabled.

Table 4-2 Required Parameters for Publisher Rabbit MQ Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`rmquserid`	Specifies the user name required to authenticate the connector's session with the Rabbit MQ server.
`rmqpassword`	Specifies the password associated with rmquserid.
`rmqhost`	Specifies the Rabbit MQ server host name.
`rmqport`	Specifies the Rabbit MQ server port.
`rmqexchange`	Specifies the Rabbit MQ exchange created by the connector, if nonexistent.
`rmqtopic`	Specifies the Rabbit MQ routing key to which messages are published.
`rmqtype`	Specifies binary, csv, json, protobuf, or opaquestring. For opaquestring, the Source window schema is assumed to be index:int64,message:string.
`urlhostport`	Specifies the *host:port* field in the metadata topic subscribed to on start-up to field metadata requests.

Table 4-3 Optional Parameters for Subscriber Rabbit MQ Connectors

Parameter	Description
collapse	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
hotfailover	Enables hot failover mode.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
buspersistence	Specify to send messages using persistent delivery mode.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when rmqtype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when rmqtype is not set to **protobuf**.
csvincludeschema	When rmqtype=CSV, specifies when to prepend output CSV data with the window's serialized schema. Valid values are never, once, and pereventblock. The default value is "never".
useclientmsgid	When performing a failover operation and extracting a message ID from an event block, use the client-generated message ID instead of the engine-generated message ID.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
rmqpasswordencrypted	Specifies that rmqpassword is encrypted.
rmqvhost	Specifies the Rabbit MQ vhost. The default is /.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.

Parameter	Description

`rmqcontenttype`	Specifies the value of the content_type parameter in messages sent to RabbitMQ. The default value depends on the message format as follows:
	• binary
	`application/octet-stream`
	• csv
	`text/csv;charset=utf-8`
	• json
	`application/json`
	• protobufs
	`application/x-protobuf`
	• opaque
	`text/csv;charset=utf-8`

`rmqheaders`	A comma–separated list of key value optional headers in messages sent to RabbitMQ. The default value is no headers.
`rmqssl`	Enables TLS encryption on the connection to the Rabbit MQ server.
`rmqsslcacert`	When rmqssl is enabled, specifies the full path of the TLS CA certificate .pem file. **Note**: You cannot use escaped characters in the path.
`rmqsslkey`	When rmqssl is enabled, specifies the full path of the TLS key .pem file. **Note**: You cannot use escaped characters in the path.
`rmqsslcert`	When rmqssl is enabled, specifies the full path of the TLS certificate .pem file. **Note**: You cannot use escaped characters in the path.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`rmqheartbeat`	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, meaning that heartbeats are disabled.
`usedeliverytagmsgid`	When extracting a message ID from an event block during a failover operation, use the delivery tag that is provided by the RabbitMQ server instead of the message ID generated by the engine.

Table 4-4 Optional Parameters for Publisher Rabbit MQ Connectors

Parameter	Description
`transactional`	When rmqtype=CSV, sets the event block type to transactional. The default event block type is normal.
`blocksize`	When rmqtype=CSV, specifies the number of events to include in a published event block. The default value is 1.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.

Parameter	Description
buspersistence	Controls both auto-delete and durable.
buspersistencequeue	Specifies the queue name used by a persistent publisher.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when rmqtype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when rmqtype is not set to **protobuf**.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window.
ackwindow	Specifies the time period (in seconds) to leave messages that are received from Rabbit MQ unacknowledged. Applies only when buspersistence = true, when, by default, messages are never acknowledged. When configured, messages are acknowledged this number of seconds after having been received. Must be configured if acktimer is configured.
acktimer	Specifies the time interval (in seconds) for how often to check whether to send acknowledgments that are triggered by the ackwindow parameter. Must be configured if ackwindow is configured.
publishwithupsert	Builds events with opcode=Upsert instead of Insert.
rmqpasswordencrypted	Specifies that rmqpassword is encrypted.
addcsvopcode	Prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
rmqvhost	Specifies the Rabbit MQ vhost. The default is /.
useclientmsgid	If the Source window has been restored from a persist to disk, ignores received binary event blocks that contain a message ID less than the greatest message ID in the restored window.
rmqssl	Enables TLS encryption on the connection to the Rabbit MQ server.
rmqsslcacert	When rmqssl is enabled, specifies the full path of the TLS CA certificate .pem file.

Parameter	Description
`rmqsslkey`	When rmqssl is enabled, specifies the full path of the TLS key .pem file.
`rmqsslcert`	When rmqssl is enabled, specifies the full path of the TLS certificate .pem file.
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events.
`rmqheartbeat`	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, meaning that heartbeats are disabled.
`usedeliverytagmsgid`	When extracting a message ID from an event block during a failover operation, use the delivery tag that is provided by the RabbitMQ server instead of the message ID generated by the engine.

4.1.2 Using the Rabbit MQ Adapter

Overview

The Rabbit MQ adapter supports publish and subscribe operations on a Rabbit MQ server. You must install the rabbitmq-c V0.9.0 client run-time libraries to use the adapter.

dfesp_rmq_adapter -C *key1=val1*,*key2=val2*,*key3=val3*,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, `key=\"str,ing\"`).

Subscriber Usage

Table 4-5 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/** **project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`rmquserid=username`	Specifies the Rabbit MQ user name.
`rmqpassword=password`	Specifies the Rabbit MQ password.
`rmqhost=hostname`	Specifies the Rabbit MQ host.
`rmqport=port`	Specifies the Rabbit MQ port.
`rmqexchange=name`	Specifies the Rabbit MQ exchange.

Key-Value Pair	Description
`rmqtopic=key`	Specifies the Rabbit MQ routing key.
`urlhostport=field`	Specifies the *host:port* field in the metadata topic to which the connector subscribes.
`numbufferedmsgs=number`	Specifies the maximum number of messages buffered by a standby subscriber connector.
`rmqtype=binary | csv | json | protobuf`	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is supported.

Table 4-6 Optional Keys

Key-Value Pairs	Description
`rmqssl=true | false`	When true, enables TLS encryption on the connection to the Rabbit MQ server.
`rmqsslcacert=path`	When rmqssl is true, specifies the full path of the TLS CA certificate .pem file. **Note**: You cannot use escaped characters in path.
`rmqsslkey=path`	When rmqssl is true, specifies the full path of the TLS key .pem file. **Note**: You cannot use escaped characters in path.
`rmqsslcert=path`	When rmqssl is true, specifies the full path of the TLS certificate .pem file. **Note**: You cannot use escaped characters in path.
`buspersistence=true | false`	When true, use durable queues and persistent messages.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when rmqtype is not set to **protobuf**.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when rmqtype is not set to **protobuf**.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.

Key-Value Pairs	Description
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`rmqpasswordencrypted=true | false`	When true, rmqpassword is encrypted.
`rmqvhost=vhost`	Specifies the Rabbit MQ virtual host. The default is /.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is ./rabbitmq.cfg. For `kafka`, the default is ./kafka.cfg. **Note**: No transport configuration file is required for native transport.
`csvincludeschema=never | once | pereventblock`	When rmqtype=CSV, specifies when to pass the window's serialized schema to subscriber callback. The default value is never.
`csvmsgperevent=true | false`	When true, for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock=true | false`	When true, for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`rmqcontenttype=value`	Specifies the value of the content_type parameter in messages sent to RabbitMQ. The default value depends on the message format, as follows: • `binary` `"application/octet-stream"` • `csv` `"text/csv;charset=utf-8"` • `json` `"application/json"` • `protobufs` `"application/x-protobuf"` • `opaque` `"text/csv;charset=utf-8"`
`rmqheaders=list`	Specifies a comma–separated list of key:value optional headers in messages sent to RabbitMQ. The default value is no headers.
`rmqheartbeat=value`	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, which means that heartbeats are disabled.

Publisher Usage

Table 4-7 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`rmquserid=username`	Specifies the Rabbit MQ user name.
`rmqpassword=password`	Specifies the Rabbit MQ password.
`rmqhost=hostname`	Specifies the Rabbit MQ host.
`rmqport=port`	Specifies the Rabbit MQ port.
`rmqexchange=name`	Specifies the Rabbit MQ exchange.
`rmqtopic=key`	Specifies the Rabbit MQ routing key.
`urlhostport=field`	Specifies the *host:port* field in the metadata topic to which the connector subscribes.
`rmqtype=binary | csv | json | prototype | opaquestringfield`	Specifies the message format. opaquestring is valid only for publishers. For opaquestring, the Source window schema is assumed to be "index:int64,message:string".

Table 4-8 Optional Keys

Key-Value Pair	Description
`rmqssl=true | false`	When true, enables TLS encryption on the connection to the Rabbit MQ server.
`rmqsslcacert=path`	When rmqssl is true, specifies the full path of the TLS CA certificate .pem file.
`buspersistence=true | false`	When true, use durable queues and persistent messages.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when rmqtype is not set to **protobuf**.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when rmqtype is not set to **protobuf**.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib`() API call.

Key-Value Pair	Description
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit`() publish/subscribe API call and in the engine `initialize`() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`rmqpasswordencrypted=true | false`	When `true`, `rmqpassword` is encrypted.
`rmqvhost=vhost`	Specifies the Rabbit MQ virtual host. The default is /.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is ./**rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. **Note**: No transport configuration file is required for native transport.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`buspersistencequeue=queue`	Specifies the queue name used by a persistent publisher.
`ackwindow=age`	Specifies the time period to leave unacknowledged messages received from Rabbit MQ when buspersistence is true.
`acktimer=seconds`	When buspersistence is true, specifies the time interval for how often to check whether to send acknowledgments that are triggered by the ackwindow parameter.
`csvfielddelimiter=delimeter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, specifies that input events are missing the key field that is autogenerated by the Source window.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`addcsvopcode= true | false`	When true, prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is true.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.

Key-Value Pair	Description
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.
`rmqheartbeat=value`	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, which means that heartbeats are disabled.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

4.2 Solace Connector and Adapter

4.2.1 Using the Solace Systems Connector

Using the Solace Systems Connector

The Solace Systems connector communicates with a hardware-based Solace fabric for publish and subscribe operations.

A Solace Systems subscriber connector receives event blocks and publishes them to this Solace topic:
*host:port/projectname/queryname/windowname/*O

A Solace Systems publisher connector reads event blocks from the following Solace topic
host:port/projectname/queryname/windowname/I
and injects them into the corresponding Source window.

As a result of the bus connectivity provided by the connector, the engine does not need to manage individual publish/subscribe connections. A high capacity of concurrent publish/subscribe connections to a single edge stream processing engine is achieved.

The Solace Systems run-time libraries must be installed on the platform that hosts the running instance of the connector. The run-time environment must define the path to those libraries (for example, specifying LD_LIBRARY_PATH on Linux platforms).

Note

To use this connector:

- Locate the reference to this connector in the `connectors: excluded`: section of the file **esp- properties.yml**.
- Set the value to `false`.

The Solace Systems connector operates as a Solace client. All Solace connectivity parameters are required as connector configuration parameters.

You must configure the following items on the Solace appliance to which the connector connects:

- a client user name and password to match the connector's `soluserid` and `solpassword` configuration parameters
- a message VPN to match the connector's `solvpn` configuration parameter
- On the message VPN, you must enable "Publish Subscription Event Messages".
- On the message VPN, you must enable "Client Commands" and "Show Commands" under "SEMP over Message Bus".☐
- On the message VPN, you must configure a nonzero "Maximum Spool Usage".
- When hot failover is enabled on subscriber connectors, you must create a single exclusive queue named "active_esp" in the message VPN. Set the queue owner to the appropriate client user name. The subscriber connector that successfully binds to this queue becomes the active connector.
- When `buspersistence` is enabled, you must enable "Publish Client Event Messages" on the message VPN.
- When `buspersistence` is enabled, you must create exclusive queues for all subscribing clients. The queue name must be equal to the `buspersistenceque` queue configured on the publisher connector (for "/I" topics), or the queue configured on the client subscriber (for "/O" topics). Add the corresponding topic to each configured queue.
- When `buspersistence` is enabled or hot failover is enabled on subscriber connectors, you must enable "Allow Guaranteed Endpoint Create", "Allow Guaranteed Message Send", and "Allow Guaranteed Message Receive" in your client profile.

When the connector starts, it subscribes to topic urlhostport/M (where urlhostport is a connector configuration parameter). This enables the connector to receive metadata requests from clients that publish or subscribe to a window in an Edge Streaming Analytics engine associated with that host:port combination. Metadata responses consist of some combination of the project, query, and window names of the window associated with the connector, as well as the serialized schema of the window.

Solace Systems subscriber connectors support a hot failover mode. The active/standby status of the connector is coordinated with the fabric so that a standby connector becomes active when the active connector fails. Several conditions must be met to guarantee successful switchovers:

- All involved connectors must be active on the same set of topics.
- All involved connectors must initiate message flow at the same time. This is required because message IDs must be synchronized across all connectors.
- Google protocol buffer support must not be enabled, because these binary messages do not contain a usable message ID.

When a new subscriber connector becomes active, outbound message flow remains synchronized due to buffering of messages by standby connectors and coordination of the resumed flow with the fabric. The size of this message buffer is a required parameter for subscriber connectors.

You can configure Solace Systems connectors to use a persistent mode of messaging instead of the default direct messaging mode. (See the description of the buspersistence configuration parameter.) This mode might require regular purging of persisted data by an administrator, if

there are no other automated mechanism to age out persisted messages. The persistent mode reduces the maximum throughput of the fabric, but it enables a publisher connector to connect to the fabric after other connectors have already processed data. The fabric updates the connector with persisted messages and synchronizes window states with the other engines in a hot failover group.

Solace Systems subscriber connectors subscribe to a special topic that enables them to be notified when a Solace client subscribes to the connector's topic. When the connector is configured with snapshot enabled, it sends a custom snapshot of the window contents to that client. This enables late subscribers to catch up upon connecting.

Solace Systems connector configuration parameters named sol… are passed unmodified to the Solace API by the connector. See your Solace documentation for more information about these parameters.

If required, you can configure the solpassword parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. When you install the Edge Streaming Analytics System Encryption and Authentication Overlay, you install the included OpenSSL executable. Use the following command on the console to invoke OpenSSL to display your encrypted password:

```
echo "solpassword" | openssl enc -e -aes-256-cbc -a -salt -pass
pass:"espSOLconnectorUsedByUser=soluserid"
```

Then copy the encrypted password into your `solpassword` parameter and enable the `solpasswordencrypted` parameter.

A Solace publisher connector instance can be configured to copy the message payload from the destination attribute instead of the body. The Source window schema must begin with an int64 key field to serve as an index that is written by the connector. To parse a message from the destination attribute, field data is pulled sequentially from each slash-separated entry and copied to Source window fields following the index field in the same order. All messages received on the corresponding topic by a connector instance that is configured to copy from the destination attribute are processed this way. Messages that still contain a body need to be received by a different connector instance and on a different topic.

Use the following parameters with Solace Systems connectors.

Table 4-9 Required Parameters for Subscriber Solace Systems Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`soluserid`	Specifies the user name required to authenticate the connector's session with the appliance.
`solpassword`	Specifies the password associated with soluserid.
`solhostport`	Specifies the appliance to connect to, in the form host:port.
`solvpn`	Specifies the appliance message VPN to assign the client to which the session connects.
`soltopic`	Specifies the Solace destination topic to which to publish.
`urlhostport`	Specifies the host:port field in the metadata topic subscribed to on start- up to field metadata requests.

Parameter	Description
numbufferedmsgs	Specifies the maximum number of messages buffered by a standby subscriber connector. If exceeded, the oldest message is discarded. If the connector goes active the buffer is flushed, and buffered messages are sent to the fabric as required to maintain message ID sequence.
snapshot	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 4-10 Required Parameters for Publisher Solace Systems Connectors

Parameter	Description
type	Specifies to publish. Must be "pub".
soluserid	Specifies the user name required to authenticate the connector's session with the appliance.
solpassword	Specifies the password associated with soluserid.
solhostport	Specifies the appliance to connect to, in the form host:port.
solvpn	Specifies the appliance message VPN to assign the client to which the session connects.
soltopic	Specifies the Solace topic to which to subscribe.
urlhostport	Specifies the host:port field in the metadata topic subscribed to on start- up to field metadata requests.

Table 4-11 Optional Parameters for Subscriber Solace Systems Connectors

Parameter	Description
collapse	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
hotfailover	Enables hot failover mode.
buspersistence	Sets the Solace message delivery mode to Guaranteed Messaging. The default delivery mode is Direct Messaging.
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
protofile	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.
protomsg	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.

Parameter	Description
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
json	Enables transport of event blocks encoded as JSON messages.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
solpasswordencrypted	Specifies that solpassword is encrypted.

Table 4-12 Optional Parameters for Publisher Solace Systems Connectors

Parameter	Description
buspersistence	Creates the Guaranteed message flow to bind to the topic endpoint provisioned on the appliance that the published Guaranteed messages are delivered and spooled to. By default this flow is disabled, because it is not required to receive messages published using Direct Messaging.
buspersistencequeue	Specifies the name of the queue to which the Guaranteed message flow binds.
protofile	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.
protomsg	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
json	Enables transport of event blocks encoded as JSON messages.
publishwithupsert	Specifies to build with opcode = Upsert instead of opcode = Insert.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.

Parameter	Description
`solpasswordencrypted`	Specifies that solpassword is encrypted.
`getmsgfromdestattr`	Extracts the payload from the destination attribute instead of the message body.
`transactional`	When getmsgfromdestattr is enabled, sets the event block type to transactional. The default event block type is normal.
`blocksize`	When getmsgfromdestattr is enabled, specifies the number of events to include in a published event block. The default value is 1.
`maxevents`	Specifies the maximum number of events to publish.

4.2.2 Using the Solace Systems Adapter

Overview

The Solace Systems adapter supports publish and subscribe operations on a hardware-based Solace fabric. You must install the Solace run-time libraries to use the adapter.

dfesp_sol_adapter -C *key1=val1,key2=val2,key3=val3,…*

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, `key=\"str,ing\")`.

Subscriber Usage

Table 4-13 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/** **project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`soluserid=username`	Specifies the Solace user name.
`solpassword=password`	Specifies the Solace password.
`solhostport=hostport`	Specifies the Solace *host:port*.
`solvpn=vpn`	Specifies the Solace VPN name.
`soltopic=topic`	Specifies the Solace topic.

Key-Value Pair	Description
`urlhostport=hostport`	Specifies the *host.port* field in the metadata topic subscribed to by the connector.
`numbufferedmsgs=number`	Specifies the maximum number of messages buffered by a standby subscriber connector.

Table 4-14 Optional Keys

Key-Value Pair	Description
`buspersistence=true | false`	When `true`, use durable queues and persistent messages.
`protofile=file`	Specifies the .proto file to be used for Google protocol buffer support.
`protomsg=message`	Specifies the message itself in the .proto file that is specified by the protofile parameter.
`json=true | false`	When true, transport JSON messages instead of event blocks.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib()` API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`solpasswordencrypted=true | false`	When true, the Solace password is encrypted.

Key-Value Pair	Description
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Publisher Usage

Table 4-15 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`soluserid=username`	Specifies the Solace user name.
`solpassword=password`	Specifies the Solace password.
`solhostport=hostport`	Specifies the Solace *host:port*.
`solvpn=vpn`	Specifies the Solace VPN name.
`soltopic=topic`	Specifies the Solace topic.
`urlhostport=hostport`	Specifies the *host:port* field in the metadata topic subscribed to by the connector.

Table 4-16 Optional Keys

Key-Value Pair	Description
`buspersistence=true | false`	When true, use durable queues and persistent messages.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter.
`json=true | false`	When true, transport JSON messages instead of event blocks.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.

Key-Value Pair	Description
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`solpasswordencrypted=true | false`	When true, the Solace password is encrypted.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. **Note**: No transport configuration file is required for native transport.
`buspersistencequeue=queue`	Specifies the queue name used by Solace Guaranteed Messaging publisher.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`getmsgfromdestattr=true | false`	When true, extracts the payload from the destination attribute instead of the message body.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

4.3 Kafka Connector and Adapter

4.3.1 Using the Kafka Connector for Kafka 0.9 and MapR Streams

Apache Kafka is an open-source streaming platform developed by the Apache Software Foundation. It is engineered to publish and subscribe to streams of records, similar to a message queue or an enterprise messaging system. It can process those streams as they occur, and then store them in a fault-tolerant way. Apache Kafka is generally used to build real-time streaming data pipelines or real-time streaming applications.

Overview

The Kafka connector communicates with a Kafka broker for publish and subscribe operations. Apache Kafka version 0.9 or higher is required. The bus connectivity provided by the connector eliminates the need for the engine to manage individual publish/subscribe connections. The connector achieves a high capacity of concurrent publish/subscribe connections to a single engine.

Note

The Kafka connector is certified to work with the MapR converged data platform for publishing and subscribing.

A Kafka subscriber connector publishes subscribed event blocks to a fixed partition in a Kafka topic. A Kafka publisher connector reads event blocks from a fixed partition in a Kafka topic and then injects the event blocks into a Source window. The topic and partition are required connector configuration parameters. You can read or write all partitions in the topic by setting the partition parameter to -1. Failover is not supported when reading orwriting all partitions in the topic, so setting the partition parameter to -1 for a single connector instance is not recommended. A better alternative is to run additional instances of the connector to read or write additional partitions in the topic.

The initial offset from which to begin consuming from a Kafka partition is an optional parameter. The default value is the smallest available offset. Using the smallest available offset ensures that state can be completely rebuilt from the topic's message store. The Kafka cluster administrator should ensure that the log.retention properties for the server are appropriately set for that offset. Other possible values for the initial offset are the largest offset, or a specific offset value.

Event blocks transmitted through a Kafka cluster can be encoded as binary, CSV, Google protobufs, JSON, or Apache Avro messages. The connector performs any conversion to and from binary format. You can configure the message format through a parameter.

The Kafka connector supports hot failover operation. When enabled for hot failover, the connector uses Apache Zookeeper to monitor the presence of other connectors in the failover group. The connector was tested using Zookeeper version 3.4.8.

The Kafka connector does not support sending custom snapshots to newly connected publish/subscribe clients that use the Edge Streaming Analytics Kafka client plug-in library. There is no need since Kafka is a message store and the initial partition offset for a client consumer is configurable in the client plug-in library.

When the connector starts, it subscribes to topic urlhostport.M (where urlhostport is a connector configuration parameter). This enables the connector to receive metadata requests from clients using the Kafka plug-in that publish or subscribe to a window in an engine associated with that host:port combination.

Remember that ':' must be replaced with '_' in the urlhostport parameter. Metadata responses consist of some combination of the following:

- project name of the window associated with the connector
- query name of the window associated with the connector
- window name of the window associated with the connector
- the serialized schema of the window

By default, all Kafka subscriber connectors and client transports require message acknowledgments from the broker. Acknowledgments enable the graceful handling of broker connection failures, topic leader changes, and producer errors signaled by the broker.

Using the Kafka Connector on Windows Systems

On Windows systems, you must install Kafka client run-time libraries on the platform that hosts the running instance of the connector. The connector uses the LibrdKafka C and C++ libraries. You can download these libraries from https://github.com/edenhill/librdkafka (https://github.com/edenhill/librdkafka).

Table 4-17 Valid Library Versions

Library Version	Edge Streaming Analytics Release
1.0.0	1.0
0.9.3	Invalid release
0.9.2	Invalid release

You must also install Zookeeper C client libraries even when hot failover operation is not enabled. You can download these libraries from http://www.apache.org/dyn/closer.cgi/zookeeper/ (http://www.apache.org/dyn/closer.cgi/zookeeper/).

Make sure the LibrdKafka and Zookeeper libraries are built on the machine where you run the connector. The ensures that the system libraries used during the build are available at run time.

To build the libraries with TLS support, on the ./configure step, use option **-LDFLAGS=-L $DFESP_HOME/lib** so that LibrdKafka is built with the same libraries as Edge Streaming Analytics.

Note

The Edge Streaming Server attempts to load the Kafka connector upon start-up. If you do not install the LibrdKafka and Zookeeper libraries, then this load fails and an error message is logged. To avoid this failure, add a reference to the Kafka connector in the connectors: excluded section of **esp-properties.yml**

Configuring LibrdKafa and Zookeeper Libraries for Use with the Kafka Connector

Except for the relevant connector parameters described in the next section, all LibrdKafka configuration parameters are left at default values by the connector. You can modify any `librdkafka` parameter from its default value via the connector `kafkaglobalconfig` and `kafkatopicconfig` parameters described below. Some latency-related parameters of interest are described at https://github.com/edenhill/librdkafka/wiki/How-to- decrease-message-latency. Steps for configuring LibrdKafka to support SASL are described at https://github.com/edenhill/librdkafka/wiki/Using-SASL-with-librdkafka.

Corresponding edge stream processing publish/subscribe client plug-in libraries are available for C and Java publish/subscribe clients. These libraries enable a standard edge stream processing publish/subscribe client application to exchange event blocks with an edge stream processing engine through a Kafka cluster. Messages are exchanged through the cluster instead of a direct TCP connection. No changes are required to the publish/ subscribe client code except for the following:

- making an additional call to `C_dfESPpubsubSetPubsubLib`() that specifies the Kafka plug-in library (for C clients)
- adding dfx-esp-kafka-api.jar to the front of your classpath (for Java clients)

The file kafka.cfg must be in the current working directory. This file specifies the client's Kafka configuration parameters.

The client plug-in libraries use fixed topic names, where these names are built from the edge stream processing URL passed by the user. The URL references the model's project and query and window names. Because Kafka has limited support for special characters, ensure that project/query/window names contain only alphanumeric characters and underscore, hyphen, and dot. To meet this requirement, the client plug-in modifies the passed URL to replace ':' with '_' and '/' with '.' when building the topic name. The configured topic in the corresponding Kafka connector must match the topic name built by the client.

When configured for hot failover operation, Zookeeper coordinates the active/standby status of the connector with other connectors running in other edge stream processing engines in the hot failover group. Thus, a standby connector becomes active when the active connector fails. All edge stream processing engines in a hot failover group must meet the following conditions to guarantee successful Switchovers:

- They must be running identical models.
- The Kafka publisher connectors must begin consuming from the same Kafka partition at the same offset. Kafka guarantees message order only within a partition. Using the same offset is required to ensure an identical state in all involved servers. In addition, message IDs added to inbound event blocks by the model must be synchronized across all publisher connectors. These message IDs also require consistent ordering. The initial offset is a required parameter for publisher connectors, which can be set to largest, smallest, or an integer number. The connector ensures that all publishers receive all messages on the partition by assigning each consumer to a different consumer group with a GUID-based unique group ID.
- The Zookeeper configuration must specify a value for tickTime no greater than 500 milliseconds. This enables subscriber connectors acting as Zookeeper clients to detect a failed client within one second, and to initiate the switch over process.

When a new subscriber connector becomes active, outbound message flow remains synchronized. Synchronization occurs because of buffering of messages by standby connectors and coordination of the resumed flow with the Kafka cluster. Specifically, the new

active subscriber connector obtains the current offset of the Kafka partition being written to, and consumes the message at current offset – 1. The active subscriber connector does the following:

- extracts the message ID from this message
- writes all buffered messages that contain a greater ID to the partition
- resumes writing messages from the subscribed window

The size of the message buffer is a required parameter for subscriber connectors.

Parameters for the Kafka Subscriber Connector

Table 4-18 Required Parameters for the Kafka Subscriber Connector

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`kafkahostport`	Specifies one or more Kafka brokers in the following form: host:port,host:port,….
`kafkatopic`	Specifies the Kafka topic.
`kafkapartition`	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled.
`kafkatype`	Specifies **binary**, **csv**, **json**, **protobuf**, **avro**, or the the name of a string field in the subscribed window schema.
`urlhostport`	Specifies the host:port field in the metadata topic that is subscribed to on start-up. This combination fields metadata requests. To meet Kafka topic naming requirements, replace ':' with '_'.
`numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector. When exceeded, the oldest message is discarded. When the connector goes active, the buffer is flushed and buffered messages are sent to the cluster as required to maintain message ID sequence.
`snapshot`	Specifies whether to send snapshot data. The only supported value is false. Subscriber clients reading messages sent by this subscriber can control their snapshot by configuring an appropriate initial offset into the Kafka partition.

Table 4-19 Optional Parameters for the Kafka Subscriber Connector

`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`hotfailover`	Enables hot failover mode.

`dateformat`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile`	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when `kafkatype` is not set to `protobuf`.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.This parameter is ignored when kafkatype is not set to protobuf.
`csvincludeschema`	When kafkatype=CSV, specifies when to prepend output CSV data with the window's serialized schema. Valid values are never, once, and pereventblock. The default value is "never".
`useclientmsgid`	Uses the client-generated message ID instead of the engine-generated message ID when performing a failover operation and extracting a message ID from an event block.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`zookeeperhostport`	Specifies the Zookeeper server in the form host:port.
`kafkaglobalconfig`	Specifies a semicolon-separated list of key=value strings to configure librdkafka global configuration values.
`kafkatopicconfig`	Specifies a semicolon-separated list of key=value strings to configure librdkafka topic configuration values.
`csvmsgperevent`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock`	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`avroschemaregistryurl`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kafkatype is not set to **avro**. See also "Publish/Subscribe API Support for Apache Avro Messaging".

avroschemadefinition	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry that is configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This definition is also copied to the schema registry. This parameter is ignored when kafkatype is not set to **avro**.
avroschemaname	Specifies the name of an Apache Avro schema to copy from the schema registry that is configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when kafkatype is not set to **avro**.
fieldtokafkakey	Specifies to copy the contents of the specified window string field to the Kafka message key. This parameter is ignored when kafkatype is not set to **protobuf**.

Parameters for the Kafka Publisher Connector

Table 4-20 Required Parameters for the Kafka Publisher Connector

Parameter	Description
type	Specifies to publish. Must be "pub".
kafkahostport	Specifies one or more Kafka brokers in the following form: host:port,host:port,….
kafkatopic	Specifies the Kafka topic.
kafkapartition	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled.
kafkatype	Specifies **binary**, **csv**, **json**, **protobuf**, **avro**, or **opaquestring**.
urlhostport	Specifies the host:port field in the metadata topic that is subscribed to on start-up. This combination fields metadata requests. To meet Kafka topic naming requirements, replace ':' with '_'.

Table 4-21 Optional Parameters for the Kafka Publisher Connector

Parameter	Description
transactional	When kafkatype=csv or kafkatype=opaquestring, sets the event block type to transactional. The default event block type is normal.
blocksize	When kafkatype=csv or kafkatype=opaquestring, specifies the number of events to include in a published event block. The default value is 1.

Parameter	Description
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`protofile`	Specifies the .proto file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.This parameter is ignored when kafkatype is not set to protobuf.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the .proto file that you specified with the protofile parameter. Event blocks are converted into this message.This parameter is ignored when kafkatype is not set to protobuf.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the ,character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window.
`publishwithupsert`	Specifies to build events with opcode = Upsert instead of opcode = Insert.
`kafkainitialoffset`	Specifies the offset from which to begin consuming messages from the Kafka topic and partition. Valid values are smallest, largest, or an integer. The default value is smallest.
`addcsvopcode`	Prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
`kafkaglobalconfig`	Specifies a semicolon-separated list of key=value strings to configure librdkafka global configuration values.
`kafkatopicconfig`	Specifies a semicolon-separated list of key=value strings to configure librdkafka topic configuration values.
`useclientmsgid`	If the Source window has been restored from a persist to disk, ignore received binary event blocks that contain a message ID less than the greatest message ID in the restored window.
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events.

Parameter	Description
`avroschemaregistryurl`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kafkatype is not set to **avro**.
`keyfromkafkakey`	Specifies to copy the Kafka message key to the window key field. This field must be type=string. This parameter is ignored when kafkatype is not set to **protobuf**.

4.3.2 Using the Kafka Adapter for Kafka 0.9 and MapR Streams

Overview

The Kafka adapter supports publish and subscribe operations on a Kafka cluster.

dfesp_kafka_adapter -C *key1=val1,key2=val2,key3=val3,…*

Note

- The Kafka adapter is certified to work with the MapR converged data platform for publishing and subscribing.
- If a value string includes a comma, then you must enclose the entire string in escaped double quotation marks (for example, key=\"str,ing\").

Subscriber Usage

Table 4-22 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append the following for subscribers:?snapshot=false. **Note:** `?snapshot=true` is invalid. When `?snapshot=true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append `?collapse=true | false` or `?rmretdel=true | false` for subscribers if needed.
`kafkahostport=broker`	Specifies one or more Kafka brokers in the following form: "host:port,host:port,...".
`kafkatopic=topic`	Specifies the Kafka topic.
`kafkapartition=partition`	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled.

Key-Value Pair	Description
`urlhostport=field`	Specifies the host:port field in the metadata topic to which the adapter subscribes (replace ':' with '_')
`numbufferedmsgs=number`	Specifies the maximum number of messages buffered by a standby subscriber connector.
`kafkatype= binary | csv | json | protobuf | avro`	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is also supported.

Table 4-23 Optional Keys

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when kafkatype is not set to **protobuf**.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when kafkatype is not set to **protobuf**.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/ subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`kafkaglobalconfig=list`	Specifies a semicolon-separated list of "key=value" strings to configure librdkafka global configuration values.
`kafkatopicconfig=list`	Specifies a semicolon-separated list of "key=value" strings to configure librdkafka topic configuration values.

Key-Value Pair	Description
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is ./solace.cfg. For `tervela`, the default is ./client.config. For `rabbitmq`, the default is ./rabbitmq.cfg. For `kafka`, the default is ./kafka.cfg. **Note**: No transport configuration file is required for native transport.
`csvincludeschema=never | once | pereventblock`	When kafkatype=CSV, specifies when to prepend output CSV data with the window's serialized schema. The default value is "never".
`csvmsgperevent= true | false`	When true, for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock=true | false`	When true, for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`avroschemaregistryurl=url`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when `kafkatype` is not set to `avro`.
`avroschemadefinition=schema`	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This schema is also copied to the schema registry. This parameter is ignored when `kafkatype` is not set to `avro`.
`avroschemaname=name`	Specifies the name of an Apache Avro schema to copy from the schema registry configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when kafkatype is not set to **avro**.
`fieldtokafkakey=fieldname`	Specifies to copy the contents of the specified window string field to the Kafka message key. This key is ignored when `kafkatype` is not set to `protobuf`.

Publisher Usage

Table 4-24 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.

Key-Value Pair	Description
`kafkahostport=broker`	Specifies one or more Kafka brokers in the following form: "host:port,host:port,...".
`kafkatopic=topic`	Specifies the Kafka topic.
`kafkapartition=partition`	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled.
`urlhostport=field`	Specifies the host:port field in the metadata topic to which the adapter subscribes (replace ':' with '_')
`kafkatype=binary | csv | json | protobuf | avro | opaquestring`	Specifies the message format. opaquestring is valid only for publishers. For opaquestring, the Source window schema is assumed to be "index:int64,message:string".

Table 4-25 Optional Keys

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=file`	Specifies the `.proto` file to be used for Google protocol buffer support.This key is ignored when `kafkatype` is not set to `protobuf`.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when `kafkatype` is not set to `protobuf`.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C+ + `C_dfESPpubsubSetPubsubLib`() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Key-Value Pair	Description
`kafkaglobalconfig=list`	Specifies a semicolon-separated list of "key=value" strings to configure librdkafka global configuration values.
`kafkatopicconfig=list`	Specifies a semicolon-separated list of "key=value" strings to configure librdkafka topic configuration values.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=delimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, specifies that input events are missing the key field that is autogenerated by the Source window.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`kafkainitialoffset=offset`	Specifies the offset from which to begin consuming from the Kafka partition. Valid values are "smallest", "largest", or an integer. The default is "smallest".
`addcsvopcode=true | false`	When true, prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is true.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.
`avroschemaregistryurl=url`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kafkatype is not set to avro.See also "Functions to Support JSON Objects".
`keyfromkafkakey=true | false`	When set to true, specifies to copy the Kafka message key to the window key field. This field must be type=string. This key is ignored when kafkatype is not set to protobuf.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

4.3.3 Failover with Kafka

Required Message Bus Configuration with Kafka

When you use a Kafka cluster in a failover topology, the Kafka connectors must have access to an Apache Zookeeper cluster. The connectors require this access in order to monitor the presence of other event stream processing servers in the failover group. When Zookeeper is installed as part of the Kafka cluster installation, you can use it for monitoring by configuring host:port on the Kafka connectors. You can download Zookeeper from https://zookeeper.apache.org (https://zookeeper.apache.org).

Note

- The Zookeeper configuration must specify a value for `tickTime` no greater than 500 milliseconds. If Zookeeper is already in use, this setting might conflict with client requirements. If there are conflicts, a separate Zookeeper installation is required for failover.
- Do not configure Kafka connectors or adapters or client transports with partition = –1. 1+N-way failover is supported only when all components read or write a single fixed partition.

Required Client Configuration with Kafka

An event stream processing publish/subscribe client application that uses Kafka requires a client configuration file named **kafka.cfg**. This configuration file must exist in the current directory to provide Kafka connectivity parameters. See the documentation of the `C_dfESPpubsubSetPubsubLib`() publish/subscribe API function for details about the contents of these configuration files.

Topic Naming with Kafka

The topic name format used on Kafka clusters is as follows:

```
host_port.project.contquery.window.direction
```

The `direction` takes the value “**I**” or “**O**”. Because this information appears in a client URL, clients can easily determine the correct appliance topic. Kafka connectors use their configured `"urlhostport"` parameter to derive the “`host:port`” section of the topic name. The rest of the information is known by the connector. To meet Kafka topic naming requirements, the configured `urlhostport` string must replace ‘:’ with ‘_’.

Determining Engine Active/Standby State with Kafka

All subscriber Kafka connectors that are enabled for failover require connectivity to a Zookeeper cluster. The first subscriber connector to start up within an event stream processing server implements a Zookeeper watcher.

When necessary, the connector also creates a root “/ESP” zNode. Then it creates a leaf node that is both sequential and ephemeral. It creates the node using the path “/ESP/server-n_seq”, where seq specifies an incrementing integer that is appended by Zookeeper. All other ESP servers in the failover group follow the same process. Thus, each server is represented in Zookeeper by a unique zNode. Each server implements a Zookeeper watcher. The first server to connect to Zookeeper has the smallest path (as identified by the seq field).

Status changes related to these Zookeeper nodes are logged to the event stream processing server console as info-level messages. When a watcher receives a zNode status change, it does the following:

- gathers all the current "**/ESP**" child nodes
- finds the path with the greatest path that is less than its own
- begins watching the zNode that owns that path

The watched zNode is the active engine. If a path was not found, the watcher itself has the smallest path and becomes the active engine.
The result is that the server with the smallest path is always the active engine. The server with the next smallest path (if there is one) is the watcher of the active engine. That server becomes the active engine when the active engine fails. In this way, no more than one zNode is watched. The zNode is watched only by one other zNode, which keeps Zookeeper chatter to a minimum. The Zookeeper `tickTime` configuration parameter must be no greater than 500 milliseconds. This enables the watcher to detect a failed active engine within one second.

New Engine Active Actions on Failover with Kafka

A standby engine server runs a Zookeeper watcher that watches the active server. When its state changes from standby to active, each subscriber Kafka connector does the following:

- queries the outbound Kafka topic for the current offset into the partition that the subscriber is configured to write to
- creates a consumer and consumes the message at `current_offset - 1` from that partition
- stops the consumer
- extracts the message ID from the event block in the received message
- begins publishing starting with the following message:
  `ID = message_ID + 1`

Suppose that the next event block produced by the subscribed window contains an ID greater than that. In that case, the connector publishes all messages in the gap from the queue that it maintained while it was in standby state. It then discards older messages from the queue. Then it resumes publishing messages produced by the subscribed window.

Metadata Exchanges with Kafka

The Kafka publish/subscribe API handles the `C_dfESPpubsubQueryMeta`(), `C_dfESPpubsubPersistModel`(), and `C_dfESPpubsubQuiesceProject`() methods as follows:

- The connectors running in a failover group listen for metadata requests on a special topic named "`urlhostport`.M". They consume the topic using a global group ID, such that only one consumer in the group processes any message sent on that topic.
- The client sends formatted messages on this topic in request-reply fashion.

- The request contains an additional field that specifies a GUID-based topic on which to send the response. Since only the requester is listening on this topic, the response is guaranteed to be received by the requester and only the requester.
- The response contains the same information that is provided by the native publish/ subscribe API.

Publisher Adapter Failover with Kafka

Failover for C and Java publisher adapters uses the Kafka failover mechanism described previously. Thus, the publisher adapter must publish event blocks to the ESP server using the Kafka transport. It also must be able to access a Kafka broker and a Zookeeper cluster. Client configuration parameters must be configured in a file named kafka.cfg.

To use the C adapter, the librdkafka and zookeeper client libraries must be installed. For more information, see “Using the Kafka Connector and Adapter (Page 72)”.

To use the Java adapter, the native Kafka Java client JAR files and Apache Zookeeper Java client JAR files must be installed. For more information, see “Using Alternative Transport Libraries for Java Clients”.

You must define the following parameters in kafka.cfg to support a failover enabled publisher adapter:

Table 4-26 Kafka Parameters

Parameter	Description
`hotfailover`	enables or disables hot failover. Valid values are **true** and **false**.
`numbufferedmsgs`	specifies the maximum number of messages to buffer on a standby adapter.
`zookeeperhostport`	specifies the Zookeeper cluster *host*:*port*.
`failovergroup`	a string defined identically for all publisher adapters that belong to the same failover group

To guarantee a successful switchover from standby adapter to active adapter, you must ensure the following:

- All publisher adapters in the failover group must be configured identically in order to ensure that they receive identical input from the publishing source. When the source is a message bus, the associated topic or queue must deliver identical messages to all clients.
- All publisher adapters in the failover group must begin publishing at approximately the same time in order to reduce buffering requirements on standby adapters. The maximum number of buffered event blocks is set by the numbufferedmsgs parameter.

Following these steps guarantees that all adapters in the failover group build the same sequence of event blocks to publish to the ESP server. It also guarantees that the unique message ID associated with each event block is assigned identically on all adapters in the group.
When you restart a failed publisher adapter, it must have access to the complete source data. It needs this access in order to get back in sync with already running adapters in the same group. If it reads incomplete data, then it assigns message IDs to event blocks incorrectly. Thus, when it becomes active, the input stream to the ESP server contains missing or duplicate event blocks.

4.4 Kinesis Connector and Adapter

4.4.1 Using the Kinesis Connector

Overview

Amazon Kinesis Data Streams is a real-time data streaming service. It is part of the Kinesis streaming data platform, along with Kinesis Data Firehose, Kinesis Video Streams, and Kinesis Data Analytics.

The Kinesis connector and adapter are Kinesis Data Streams applications that enable you to collect and process large streams of data records in real time. You can use them to send edge streaming analytics data to a variety of other AWS services by managing your Kinesis Data Stream in the AWS console.

Using the Kinesis Connector

The Kinesis connector connects to a Kinesis stream for publish or subscribe operations. This connectivity eliminates the need for the engine to manage individual publish/subscribe connections. The connector achieves a high capacity of concurrent publish/subscribe connections to a single engine.

The connector supports a pair of optional authentication parameters (awsaccesskeyid and awssecretkey), both of which must be configured to provide explicit credentials to the connector. If these are not configured, then the connector uses one of several other authentication methods described on the AWS web site.

By default, a Kinesis subscriber connector publishes subscribed event blocks to a fixed shard in a Kinesis stream. Alternatively, a subscriber connector can write to a configurable number of shards in the stream. partitionkey is a required parameter, and combined with the optional partitionkeyrange parameter, you can define a set of partition keys to write to the stream. Each resulting partition key is used as input to a hash function that maps the key to a specific shard per output event block. If you do not configure partitionkeyrange, the same partition key is used for all output records. Generally, the number of partition keys should be much larger than the number of shards configured on the stream, to reduce latency and maximize throughtput. Be aware that Amazon imposes a 1MB limit per written record.

A Kinesis publisher connector reads records from a configurable set of shards in a Kinesis stream and then injects the event blocks into a source window. It runs a separate thread per configured shard to optimize throughput. By default, it reads all available records per `GetRecords`() operation, but you can configure this number with the `maxrecordspershard` parameter. You might need to adjust the number to conform to the read throughput per shard supported by your stream. `sharditeratortype` is a required publisher parameter that controls where to begin reading from the stream.

Records transmitted through a Kinesis stream can be encoded as binary, CSV, Google protobufs, JSON, or Apache Avro messages. The connector performs any conversion to and

from internal binary format. You can configure the message format using the `kinesistype` parameter.

Table 4-27 Required Parameters for the Kinesis Subscriber Connector

Parameter	Description
`type`	Specifies to subscribe. Must be **sub**.
`kinesistype`	Specifies **binary**, **csv**, **json**, **protobuf**, **avro**, or the name of a string field in the subscribed window schema.
`snapshot`	Specifies whether to send snapshot data. When **true**, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.
`streamname`	Specifies the Amazon Kinesis Data Stream name.
`awsregion`	Specifies the Amazon AWS region.
`partitionkey`	Specifies a string used to map output records to a specific shard. Configure `partitionkeyrange` to write to multiple shards.

Table 4-28 Required Parameters for the Kinesis Publisher Connector

Parameter	Description
`type`	Specifies to publish. Must be **pub**.
`kinesistype`	Specifies **binary**, **csv**, **json**, **protobuf**, **avro**, or **opaquestring**.
`streamname`	Specifies the Amazon Kinesis Data Stream name.
`awsregion`	Specifies the Amazon AWS region.
`sharditeratortype`	Valid values are **trimhorizon**, **latest**, **atsequencenumber**, **aftersequencenumber**, or **attimestamp**.

Table 4-29 Optional Parameters for the Kinesis Subscriber Connector

Parameter	Description
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.

Parameter	Description
`protofile`	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when kinesistype is not set to **protobuf**.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when kinesistype is not set to **protobuf**.
`csvincludeschema`	When kinesistype=csv, specifies when to prepend output CSV data with the window's serialized schema. Valid values are **never**, **once**, and **pereventblock**. The default value is **never**.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters.
`csvmsgperevent`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock`	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`avroschemaregistryurl`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kinesistype is not set to **avro**. See also "Functions to Support Apache Avro Objects".
`avroschemadefinition`	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry that is configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This definition is also copied to the schema registry. This parameter is ignored when `kinesistype` is not set to **avro**.
`avroschemaname`	Specifies the name of an Apache Avro schema to copy from the schema registry that is configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when `kinesistype` is not set to **avro**.
`awsaccesskeyid`	Specifies the AWS access key id credential. Required when `awssecretkey` is configured.
`awssecretkey`	Specifies the AWS secret key credential. Required when `awsaccesskeyid` is configured.
`partitionkeyrange`	Specifies the number of partition keys used when writing output records. The default value is 1.

Table 4-30 Optional Parameters for the Kinesis Publisher Connector

Parameter	Description
`transactional`	When kinesistype=csv, sets the event block type to transactional. The default event block type is normal.
`blocksize`	When kinesistype=csv, specifies the number of events to include in a published event block. The default value is 1.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`protofile`	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when kinesistype is not set to **protobuf**.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when kinesistype is not set to **protobuf**.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters.
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the **,** character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window.
`publishwithupsert`	Specifies to build events with `opcode = Upsert` instead of `opcode = Insert`.
`addcsvopcode`	Prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events
`avroschemaregistryurl`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when `kinesistype` is not set to **avro**. See also "Functions to Support Apache Avro Objects"
`awsaccesskeyid`	Specifies the AWS access key id credential. Required when `awssecretkey` is configured.
`awssecretkey`	Specifies the AWS secret key credential. Required when `awsaccesskeyid` is configured.

Parameter	Description
`shardids`	Specifies a comma separated list of shard IDs to read from. The default value is to read from all shards in the stream.
`shardsequencenumber`	Specifies the sequence number to use when `sharditeratortype` is set `toatsequencenumber` or `aftersequencenumber`.
`shardtimestamp`	Specifies the timestamp to use when `sharditeratortype` is set to `attimestamp`. See the `dateformat` parameter description for the required format.
`maxrecordspershard`	Specifies the maximum number of records to read from the shard per `GetRecords`() call. The default value is to read all available records.

4.4.2 Using the Kinesis Adapter

Overview

The Kinesis adapter supports publish and subscribe operations through a Kinesis Data Streams stream.

dfesp_kinesis_adapter -C *key1*=*val1*,*key2*=*val2*,*key3*=*val3*,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotation marks (for example, **key=\"str,ing\"**).

Subscriber Usage

Table 4-31 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append the following for subscribers: **?snapshot=true | false**. When **? snapshot=true**, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append **?collapse=true | false** or **?rmretdel=true | false** for subscribers if needed.
`kinesistype= binary | csv | json | protobuf | avro`	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is also supported.
`streamname=name`	Specifies the Amazon Kinesis Data Stream name.

Key-Value Pair	Description
`awsregion=region`	Specifies the Amazon AWS region.
`partitionkey=key`	Specifies a string used to map output records to a specific shard. Configure `partitionkeyrange` to write to multiple shards.

Table 4-32 Optional Keys

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=file`	Specifies the .proto file to be used for Google protocol buffer support. This key is ignored when `kinesistype` is not set to `protobuf`.
`protomsg=message`	Specifies the message itself in the .proto file that is specified by the protofile parameter. This key is ignored when kinesistype is not set to protobuf.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[secti on]`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For Solace, the default is **./solace.cfg**. For Tervela, the default is **./client.config**. For RabbitMQ, the default is **./rabbitmq.cfg**. For Kafka, the default is ./kafka.cfg. **Note:** No transport configuration file is required for native transport.
`csvincludeschema=never | once | pereventblock`	When kinesistype=csv, specifies when to prepend output CSV data with the window's serialized schema. The default value is never.
`csvmsgperevent= true | false`	When true, for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock=true | false`	When true, for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`avroschemaregistryurl=url`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kinesistype is not set to **avro**. See also "Functions to Support Apache Avro Objects".
`awsaccesskeyid=id`	Specifies the AWS access key ID credential. Required when awssecretkey is configured.
`awssecretkey=key`	Specifies the AWS secret key credential. Required when awsaccesskeyid is configured.
`avroschemadefinition=schema`	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This schema is also copied to the schema registry. This parameter is ignored when kinesistype is not set to **avro**.
`avroschemaname=name`	Specifies the name of an Apache Avro schema to copy from the schema registry configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when kinesistype is not set to **avro**.
`partitionkeyrange=number`	Specifies the number of partition keys used when writing output records. The default value is 1.

Publisher Usage

Table 4-33 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.

Key-Value Pair	Description
`kinesistype=binary | csv | json |protobuf | avro | opaquestring`	Specifies the message format. **opaquestring** is valid only for publishers. For **opaquestring**, the source window schema is assumed to be "index:int64,message:string".
`streamname=name`	Specifies the Amazon Kinesis Data Stream name.
`awsregion=region`	Specifies the Amazon AWS region.
`sharditeratortype=trimho rizon | latest | atsequencenumber | aftersequencenumber | attimestamp`	Specifies the Amazon Kinesis shard iterator type.

Table 4-34 Optional Keys

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the .proto file to be used for Google protocol buffer support. This key is ignored when kinesistype is not set to protobuf.
`protomsg=message`	Specifies the message itself in the .proto file that is specified by the protofile parameter. This key is ignored when kinesistype is not set to protobuf.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[secti on]`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters.
`restartonerror=true | false`	When **true**, specifies to restart the adapter if a fatal error is reported.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For Solace, the default is **./solace.cfg**. For Tervela, the default is **./client.config**. For RabbitMQ, the default is **./rabbitmq.cfg**. For Kafka, the default is ./kafka.cfg. **Note:** No transport configuration file is required for native transport.
`awsaccesskeyid=id`	Specifies the AWS access key ID credential. Required when awssecretkey is configured.
`awssecretkey=key`	Specifies the AWS secret key credential. Required when awsaccesskeyid is configured.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=delimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the **,** character.
`noautogenfield=true | false`	When **true**, specifies that input events are missing the key field that is autogenerated by the Source window.
`publishwithupsert=true | false`	When **true**, build events with opcode = Upsert instead of Insert.
`addcsvopcode=true | false`	When **true**, prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is **true**.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to insert into input CSV events (with comma).
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When **true**, quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.
`avroschemaregistryurl=url`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kinesistype is not set to avro. See also "Functions to Support Apache Avro Objects".
`shardids=list`	Specifies a comma separated list of shard IDs to read from. The default value is to read from all shards in the stream.
`shardsequencenumber=number`	Specifies the sequence number to use when sharditeratortype is set to **atsequencenumber** or **aftersequencenumber**.
`shardtimestamp=timestamp`	Specifies the timestamp to use when sharditeratortype is set to **attimestamp**. See the dateformat parameter description for the required format.
`maxrecordspershard=number`	Specifies the maximum number of records to read from the shard per `GetRecords`() call. The default value is to read all available records.

4.5 Tibco Rendezvous Connector and Adapter

4.5.1 Using the Tibco Rendezvous (RV) Connector

Overview

Tibco Rendezvous (RV) is a software product that provides a message bus for enterprise application integration (EAI). It includes an API and the Tibco RV daemon. The daemon is a background process that orchestrates data transport across computer systems.

Using the Tibco Rendezvous (RV) Connector

The Tibco Rendezvous (RV) connector supports the Tibco RV API for publish and subscribe operations through a Tibco RV daemon. The subscriber receives event blocks and publishes them to a Tibco RV subject. The publisher is a Tibco RV subscriber, which injects received event blocks into source windows.

The Tibco RV run-time libraries must be installed on the platform that hosts the running instance of the connector. The run-time environment must define the path to those libraries (for example, specifying LD_LIBRARY_PATH on Linux platforms).

Note

To use this connector:

- Locate the reference to this connector in the `connectors: excluded`: section of the file esp- properties.yml.
- Set the value to `false`.

The system path must point to the **Tibco/RV/bin** directory so that the connector can run the RVD daemon.

The subject name used by a Tibco RV connector is a required connector parameter. A Tibco RV subscriber also requires a parameter that defines the message format used to publish events to Tibco RV. A Tibco RV publisher can consume any message type produced by a Tibco RV subscriber.

By default, the Tibco RV connector assumes that a Tibco RV daemon is running on the same platform as the connector. Alternatively, you can specify the connector tibrvdaemon configuration parameter to use a remote daemon.

Similarly, you can specify the optional `tibrvservice` and `tibrvnetwork` parameters to control the Rendezvous service and network interface used by the connector. For more information, see your Tibco RV documentation.

The Tibco RV connector relies on the default multicast protocols for message delivery. The reliability interval for messages sent to and from the Tibco RV daemon is inherited from the value in use by the daemon.

Use the following parameters with Tibco RV connectors:

Table 4-35 Required Parameters for Subscriber Tibco RV Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`tibrvsubject`	Specifies the Tibco RV subject name.
`tibrvtype`	Specifies **binary**, **csv**, **json**, **protobuf**, or the name of a string field in the subscribed window schema.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 4-36 Required Parameters for Publisher Tibco RV Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`tibrvsubject`	Specifies the Tibco RV subject name.
`tibrvtype`	Specifies **binary**, **csv**, **json**, **protobuf**, or **opaquestring**. For **opaquestring**, the Source window schema is assumed to be index:int64,message:string.

Table 4-37 Optional Parameters for Subscriber Tibco RV Connectors

Parameter	Description
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`tibrvservice`	Specifies the Rendezvous service used by the Tibco RV transport created by the connector. The default service name is "rendezvous".
`tibrvnetwork`	Specifies the network interface used by the Tibco RV transport created by the connector. The default network depends on the type of daemon used by the connector.
`tibrvdaemon`	Specifies the Rendezvous daemon used by the connector. The default is the default socket created by the local daemon.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.

Parameter	Description
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when tibrvtype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when tibrvtype is not set to **protobuf**.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
Csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Table 4-38 Optional Parameters for Publisher Tibco RV Connectors

Parameter	Description
blocksize	Specifies the number of events to include in a published event block. The default value is 1.
transactional	Sets the event block type to transactional. The default event block type is normal.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
tibrvservice	Specifies the Rendezvous service used by the Tibco RV transport created by the connector. The default service name is rendezvous.
tibrvnetwork	Specifies the network interface used by the Tibco RV transport created by the connector. The default network depends on the type of daemon used by the connector.
tibrvdaemon	Specifies the Rendezvous daemon used by the connector. The default is the default socket created by the local daemon.

Parameter	Description
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or %**ProgramData\Mdsp\esa\EdgeStreamingServer\default\connectors**.config (Windows) to parse for configuration parameters. Specify the value as [`configfilesection`].
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`protofile`	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when tibrvtype is not set to **protobuf**.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when tibrvtype is not set to **protobuf**.
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window.
`publishwithupsert`	Specifies to build events with opcode=Upsert instead of opcode=Insert.
`addcsvopcode`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events.

4.5.2 Using the Tibco Rendezvous (RV) Adapter

Overview

The Tibco RV adapter supports publish and subscribe operations using a Tibco RV daemon. You must install the Tibco RV run-time libraries to use the adapter.

dfesp_tibrv_adapter -C *key1=val1,key2=val2,key3=val3,…*

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Subscriber Usage

Table 4-39 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`tibrvsubject=subject`	Specifies the Tibco RV subject. Rendezvous programs communicate by sending messages. Each message bears a subject name.
`tibrvtype=binary | csv | json | protobuf`	Specifies a message form of binary, csv, json or protobuf. For subscribers, the name of a string field in the subscribed window schema is also supported.

Table 4-40 Optional Keys

Key-Value Pair	Description
`tibrvservice=service`	Specifies the Tibco RV service. Rendezvous daemon processes communicate using UDP or PGM services.
`tibrvnetwork=network`	Specifies the Tibco RV network.
`tibrvdaemon=daemon`	Specifies the Tibco RV daemon.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when `tibrvtype` is not set to **protobuf**.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when `tibrvtype` is not set to **protobuf**.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib`() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.

Key-Value Pair	Description
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [`configfilesection`].
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. **Note**: No transport configuration file is required for native transport.
`csvmsgperevent=true | false`	When true, send one message per event. By default, one message is sent per transactional event block.
`csvmsgpereventblock=true | false`	When true, send one message per event block. By default, one message is sent per transactional event block.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Publisher Usage

Table 4-41 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**.
`tibrvsubject=subject`	Specifies the Tibco RV subject.
`tibrvtype=binary | csv | json | protobuf | opaquestring`	Specifies a message format of `binary, csv, json`, or `protobuf`. `opaquestring` is supported for publishers. For opaquestring, the Source window schema is assumed to be "`index:int64,message:string`".

Table 4-42 Optional Keys

Key-Value Pair	Description
`tibrvservice=service`	Specifies the Tibco RV service.
`tibrvnetwork=network`	Specifies the Tibco RV network.
`tibrvdaemon=daemon`	Specifies the Tibco RV daemon.

Key-Value Pair	Description
`dateformat=format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (STAMP) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=file`	Specifies the .**proto** file to be used for Google protocol buffer support.This key is ignored when `tibrvtype` is not set to `protobuf`.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib`() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=delimeter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, specifies that input events are missing the key field that is autogenerated by the Source window.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.

Key-Value Pair	Description
`addcsvopcode= true | false`	When true, prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is true.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When `true`, strip all nulls from input CSV events.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

Other protocols for Edge Streaming Server 5

5.1 Using the URL Connector

The URL connector enables you to bring data into Edge Streaming Analytics over a URL-based connection, transform the data, and publish events into an edge streaming analytics model. Some examples of data that you can publish into Edge Streaming Analytics with a URL connector are:

- XML data pulled from several RSS headline news feeds
- Live weather data in JSON pulled with REST requests from a weather service
- News stories pulled from an HTML page

A single URL request can generate many events. The URL connector uses event loop technology to parse, transform, and publish the data into an Edge Streaming Server as events.

The XML defining the edge streaming analytics model that uses the URL connector points to an external configuration file. This configuration file contains information about the URLs to retrieve and how to transform the information from each URL into data that can be published into an Edge Streaming Server.

For sample configuration files, see the URL connector examples located in the examples folder of your Edge Streaming Analytics installation directory:

(Linux) **$DFESP_HOME/examples/xml/url_connector_news/config.xml or DFESP_HOME/examples/xml/url_connector_weather/config.xml**

(Windows) **DFESP_HOME\examples\xml\url_connector_news\config.xml or DFESP_HOME\examples\xml\url_connector_weather\config.xml**

The URL connector enables you to specify any number of publishers for the associated Source window. Each publisher has its own transformation rules to prepare the data for edge stream processing publishing. You can define any number of requests for each publisher. Requests can reside in the same publisher if they use the same data preparation rules to form events. For example, you can have a single publisher send REST requests to several news feeds. You then can publish the feeds into an Edge Streaming Server using a single publisher and several request definitions.

Use the URL connector only to publish events.

Table 5-1 Required Parameters for Publisher URL Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`configUrl`	Specifies the URL for the connector configuration.

Table 5-2 Optional Parameters for Publisher URL Connectors

Parameter	Description
`interval`	Specifies the interval at which the requests are sent. The default is 10 seconds.
`properties`	Specifies the properties that can be used in the configuration. The properties are entered as a semicolon- delimited list of name-value pairs.
`maxevents`	Specifies the maximum number of events to publish.

5.2 Timer Connector and Adapter

5.2.1 Using the Timer Publisher Connector

The timer publisher connector generates and publishes trigger events at regular intervals. On every interval expiration, an event with schema <id*:int64,time:stamp,label:string> is generated. The connector's Source window must follow this schema.

The timer interval is defined as follows, as specified by connector parameters:

- A start time
- An interval that could be any number of seconds, minutes, hours, days, weeks, months, or years
- A label string that identifies the timer source, in case there are multiple timer connector instances. If not specified, the default is the connector's name.

The start time can be a date in the future or in the past. If it is in the future, this is a start time; if it is in the past, this is more of a base time.

Suppose the time is set to 2017-08-01 11:25:32 and the interval to 1 hour. If the current time when the model starts is 2017-09-15 01:12:31, then the first event is generated at 2017-09-15 01:25:32, the second at 2017-09-15 02:25:32, and so on.

Table 5-3 Required Parameters for Timer Connectors

Parameter	Description
`type`	Specifies to publish or subscribe.
`basetime`	Specifies the start time in the format defined by the timeformat parameter.
`interval`	Specifies the interval length in units defined by the unit parameter.
`unit`	Specifies the unit of the interval parameter. Units include second | minute | hour | day | week | month | year.

Table 5-4 Optional Parameters for Timer Connectors

Parameter	Description
`label`	Specifies the string to be written to the source window "label" field. The default value is the connector name.
`timeformat`	Specifies the format of the basetime parameter. The default value is "%Y-%m-%d %H:%M:%S".
`transactional`	Sets the event block type to transactional. The default value is normal.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`publishwithupsert`	Builds events with opcode = Upsert instead of opcode = Insert.
`maxevents`	Specifies the maximum number of events to publish.

5.2.2 Using the Timer Publisher Adapter

The timer publisher adapter provides the same functionality as the timer publisher connector, with the addition of some adapter-only optional parameters.

dfesp_timer_adapter -C *key1=val1*,*key2=val2*,*key3=val3*,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 5-5 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**.
`basetime=time`	Specifies the start time in the format defined by the timeformat parameter.
`interval=length`	Specifies the interval length in units defined by the unit parameter.
`unit=second | minute | hour | day | week | month | year`	Specifies the units of the interval parameter.

Table 5-6 Optional Keys

Key-Value Pair	Description
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`transactional=true | false`	When true, events are transactional.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`label=string`	Specifies the string to be written to the Source window label field. The default value is the connector name.
`timeformat=format`	Specifies the format of the basetime parameter. The default value is %Y-%m-%d %H:%M:%S.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.

5.3 Pylon Publisher Connector and Adapter

5.3.1 Using the Pylon Publisher Connector

The Pylon connector communicates with a Basler GigE camera to continuously publish captured frames into a Source window. The camera must have a known, fixed IP address, and the attached Ethernet network cable must provide power using Power-over-Ethernet.

If the camera does not have an IP configuration or its IP address is unknown, you must run the Pylon IP Configurator. The Pylon IP Configurator can be found in the Basler Pylon Camera Software Suite available for download at https://www.baslerweb.com/en/support/downloads/software-downloads/. In order for the camera to be discovered, it must be connected to the same subnet as the machine running the software. You can then copy its IP address into the connector configuration to run the connector. If no camera IP address has been configured, the connector connects to the first camera found on the local subnet. Running the connector without a camera IP address might be useful for testing, but it is not recommended for production deployments.

To optimize performance, configure all network interfaces between the camera and the machine running the connector to support jumbo frames. Be sure to configure the camerapacketsize connector parameter with the same MTU value.

The frame data received by the connector is uncompressed and copied into a Source window field of type ESP_BINARY. The Source window schema must consist of a 64–bit integer followed by the binary blob field (for example: id*:int64,frame:blob). The integer field, id, is incremented by the connector with every published event and serves as the window key field.

By default, frames are published as fast as the camera can provide them, but the user can configure the connector to publish frames at a slower, fixed rate. Basic frame parameters such as pixel format and Area-Of- Interest width and height are configurable with connector configuration parameters.

Full access to the complete set of camera configuration parameters is available only in the Basler Pylon Camera Software Suite mentioned earlier. This utility can save a camera's configuration to a file, and the path to that file can be configured on the connector to allow the user to run the camera with any valid camera configuration.

Multiple connector instances can be started to capture frames simultaneously from different cameras and publish them to any combination of Source windows.

Note

You must install the Pylon run-time libraries on the platform that hosts the running instance of the connector. These libraries are installed as part of the Pylon Camera Software Suite installation and can be found under the same **/pylon*** directory. The run-time environment must define the path to those libraries (for example, specifying LD_LIBRARY_PATH on Linux platforms).

Table 5-7 Required Parameters for Pylon Connectors

Parameter	Description
`type`	Specifies to publish. Must be pub.

Table 5-8 Optional Parameters for Pylon Connectors

Parameter	Description
`cameraipaddress`	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet.
`maxnumframes`	Specifies the maximum number of frames to publish. The default value is no maximum.
`maxframerate`	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
`camerafeaturesfile`	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
`camerawidth`	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
`cameraheight`	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
`camerapixelformat`	Specifies the image pixel format. The default value is the format in the current camera configuration.
`camerapacketsize`	Specifies the Ethernet packet size. The default value is the value in the current camera configuration.
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/ config/ etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`publishwithupsert`	Builds events with opcode=Upsert instead of opcode=Insert.
`cameraxoffset`	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
`camerayoffset`	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.
`maxevents`	Specifies the maximum number of events to publish.

5.3.2 Using the Pylon Publisher Adapter

The Pylon publisher adapter provides the same functionality as the Pylon publisher connector, with the addition of some adapter-only optional parameters.

dfesp_pylon_adapter -C *key1*=*val1*,*key2*=*val2*,*key3*=*val3*,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 5-9 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.

Table 5-10 Optional Keys

Key-Value Pair	Description
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`transactional=true | false`	When true, events are transactional.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`cameraipaddress=ipaddress`	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet.
`maxnumframes=number`	Specifies the maximum number of frames to publish. The default value is no maximum.
`maxframerate=number`	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
`camerafeaturesfile=configfile`	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
`camerawidth=width`	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
`cameraheight=height`	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
`camerapixelformat=format`	Specifies the image pixel format. The default value is the format in the current camera configuration.
`camerapacketsize=size`	Specifies the Ethernet packet size. The default value is the value in the current camera configuration.
`cameraxoffset=horizontaloffset`	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
`camerayoffset=verticaloffset`	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.
`restartonerror=true | false`	When true, specifies to restart the adapter when a fatal error is reported.

5.4 UVC Connector and Adapter

5.4.1 Using the UVC Connector

The UVC connector enables you to publish photos taken by a V4L2–compatible camera to Edge Streaming Analytics. One UVC connector can run in memory at a time. A camera's streaming speed depends on several factors, including the following:

- Resolution of the streaming photos
- The format of the image

- The quality of the camera
- The value of the frame_rate adapter configuration parameter

You can specify image format, frame rate, image size, and other photo properties in the UVC connector parameters of a model's XML definition. When the UVC connector starts, the console returns a list of the image formats, frame rates, and image sizes that the camera supports.

The UVC connector supports MJPG (Motion-JPEG) and YUYV. If the format_in or format_out connector parameters specify a format other than MJPG and YUYV or if the camera does not support the format that is specified, the connector fails.

Note

MJPG format might miss some segments that exist in JPEG format. Edge Streaming Analytics detects whether a DHT (Define Huffman Table) exists. If the table does not exist, the default DHT is added to the image in order to make the output valid JPEG.

If the width and height parameters specify an image size that is not supported by the camera, the camera scales the photo to the closest compatible size. Then the console issues a warning about the difference in captured and published image sizes and the process continues. Similarly, if the frame_rate parameter specifies a frame rate that is not supported by the camera, the connector sets the camera's capture rate to the closest supported rate.

Note

The frame_rate parameter specifies the published frame rate, which is distinct from the captured frame rate. The captured frame rate is the frame rate at which the camera captures photos. This rate might not be the same as the published frame rate that is specified by the frame_rate parameter.

The connector attempts to set a camera's capture frame rate to the same value as the publish frame rate. When a camera does not support the publish frame rate, the connector might set the camera's capture frame rate to a faster or slower rate. When the rate is faster, an old frame that has not been published might be dropped and replaced by a new frame. When the rate is slower, a frame might be published more than once until a new frame arrives.

The UVC connector can be used only to publish events.

Table 5-11 Required Parameters for the UVC Publisher Connector

Parameter	Description
`type`	Specifies to publish. Must be "pub".

Table 5-12 Optional Parameters for the UVC Publisher Connector

Parameter	Description
`frame_rate`	Specifies the frames per second that the camera streams. Must be a double. The default value is 15.
`format_in`	Specifies the image format of captured photos. The default is mjpg". yuyv, an uncompressed image format, is also supported.
`format_out`	Specifies the image format that the connector publishes. The default is mjpg. yuyv, an uncompressed image format, is supported only when format_in is yuyv.
`width`	Specifies the width of the photo. Not all resolutions are supported. Consult the camera menu or use the v4l2-ctl command-line utility for a list of supported resolutions.
`height`	Specifies the height of the photo. Not all resolutions are supported. Consult the camera menu or use the v4l2-ctl command-line utility for a list of supported resolutions.
`brightness`	Specifies the brightness of the photo. Consult the camera menu or use the v4l2-ctl command- line utility for a list of supported brightness values.
`gain`	Specifies the gain of the photo. Consult the camera menu or use the v4l2-ctl command- line utility for a list of supported gain values.
`saturation`	Specifies the saturation of the photo. Consult the camera menu or use the v4l2-ctl command- line utility for a list of supported saturation values.
`contrast`	Specifies the contrast of the photo. Consult the camera menu or use the v4l2-ctl command- line utility for a list of supported contrast values.
`device`	Specifies the device name the camera is using on the Linux operating system. The device name is typically **/dev/videoX**.
`predelay`	Specifies a delay time, in seconds, on starting the connector. In some environments, the camera might take time to reset and initialize. If the camera fails to start and predelay is n, the connector waits n seconds to start. If the connector fails to detect the camera, the Edge Streaming Server exits with a fatal error.
`maxevents`	Specifies the maximum number of events to publish.

5.4.2 Using the UVC Publisher Adapter

The UVC adapter enables you to publish photos taken by a V4L2–compatible camera to Edge Streaming Analytics.

dfesp_uvc_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 5-13 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**.

Table 5-14 Optional Keys

Key-Value Pair	Description
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`frame_rate=number`	Specifies the number of frames per second. The default is 15.
`device=name`	Specifies the device name. The default is "/dev/video0".
`format_in=format`	Specifies the image format of captured photos. The default is mjpg. yuyv, an uncompressed image format, is also supported.

Key-Value Pair	Description
`format_out=format`	Specifies the photo format output from the adapter. The default is mjpg. yuyv is an uncompressed image format. It is supported only when format_in is yuyv
`width=width`	Specifies the width for the frame resolution. The default is 640.
`height=height`	Specifies the height for the frame resolution. The default is 480.
`brightness=level`	Specifies the brightness level. The default value is 0. Consult the camera menu or use the **v4l2-ctl** command-line utility for a list of supported brightness values.
`gain=gain`	Specifies the gain. The default value is 0. Consult the camera menu or use the **v4l2-ctl** command-line utility for a list of supported gain values.
`saturation=saturation`	Specifies the saturation. The default value is 0. Consult the camera menu or use the **v4l2-ctl** command-line utility for a list of supported saturation values.
`contrast=contrast`	Specifies the contrast. The default value is 0. Consult the camera menu or use the **v4l2-ctl** command-line utility for a list of supported contrast values.
`predelay=seconds`	Specifies the number of seconds to delay before a retry if unable to start camera. The default is 0.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.

5.5 File and Socket Connector and Adapter

5.5.1 Using File and Socket Connectors

File and socket connectors support both publish and subscribe operations on files or socket connections that stream the following data types:

- binary
- cef (ArcSight Common Event Format, supports only publish operations)
- csv
- hdat (supports only subscribe operations)

- json
- syslog (supports only publish operations)
- xml

The file or socket nature of the connector is specified by the form of the configured fsname. A name in the form of *host*: *port* is a socket connector. Otherwise, it is a file connector.

When the connector implements a socket connection, it might act as a client or server, regardless of whether it is a publisher or subscriber. When you specify both *host* and *port* in the **fsname**, the connector implements the client. The configured host and port specify the network peer implementing the server side of the connection.

However, when *host* is blank (that is, when **fsname** is in the form of ": *port*"), the connection is reversed. The connector implements the server and the network peer is the client.

Use the following parameters when you specify file and socket connectors.

Table 5-15 Required Parameters for File and Socket Connectors

Parameter	Description
`fsname`	Specifies the input file for publishers, output file for subscribers, or socket connection in the form of *host*: *port*. Leave *host* blank to implement a server instead of a client.
`fstype`	Specifies the type of file: binary | csv | xml | json | syslog | hdat | cef.
`type`	Specifies whether to publish or subscribe.

Table 5-16 Required Parameters for Subscriber File and Socket Connectors

Parameter	Description
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 5-17 Optional Parameters for Subscriber File and Socket Connectors

Parameter	Description
`collapse`	Converts UPDATE_BLOCK events to UPDATE events in order to make subscriber output publishable.
`configfilesection`	Specifies the name of the section in /opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Parameter	Description
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`hdatcashostport`	Specifies the CAS server host and port. Applies only to the hdat connector.
`hdatcaspassword`	Specifies the CAS server password. Applies only to the hdat connector.
`hdatcasusername`	Specifies the CAS server user name. Applies only to the hdat connector.
`hdatfilename`	Specifies the name of the Objective Analysis Package Data (HDAT) file to be written to the Hadoop Distributed File System (HDFS). Include the full path, as shown in the HDFS browser when you browse the name node specified in the fsname parameter. Do not include the .sashdat extension. Applies only to and is required for the **hdat** connector.
`hdatlasrhostport`	Specifies the SAS LASR Analytic Server host and port. Applies only to the **hdat** connector.
`hdatlasrkey`	Specifies the path to **tklasrkey.sh**. By default, this file is located in **$DFESP_HOME/bin**. Applies only to the **hdat** connector.
`hdatmaxdatanodes`	Specifies the maximum number of data node connections. The default value is the total number of live data nodes known by the name node. This parameter is ignored when hdatnumthreads <=1. Applies only to the **hdat** connector.
`hdatmaxstringlength`	Specifies in bytes the fixed size of string fields in Objective Analysis Package Data (HDAT) files. You must specify a multiple of 8. Strings are padded with spaces. Applies only to and is required for the **hdat** connector. To specify unique string lengths per column, configure a comma-separated string of values, using no spaces. Every value must be a multiple of 8, and the number of values must be equal to the number of string fields in the subscribed window schema.
`hdatnumthreads`	Specifies the size of the thread pool used for multi-threaded writes to data node socket connections. A value of 0 or 1 indicates that writes are single- threaded and use only a name-node connection. Applies only to and is required for the **hdat** connector.
`hdfsblocksize`	Specifies in Mbytes the block size used to write an Objective Analysis Package Data (HDAT) file. Applies only to and is required for the **hdat** connector.
`hdfsnumreplicas`	Specifies the number of Hadoop Distributed File System (HDFS) replicas created with writing an Objective Analysis Package Data (HDAT) file. The default value is 1. Applies only to the **hdat** connector.

Parameter	Description
`header`	false | true | full For a CSV subscriber, specifies to write a header row that shows comma- separated fields. Set the value to full to include opcode, flags, in the header line.
`maxfilesize`	Specifies the maximum size in bytes of the subscriber output file. When reached, a new output file is opened. When configured, a timestamp is appended to all output filenames. This parameter does not apply to socket connectors with fstype other than **hdat**.
`periodicity`	Specifies the interval in seconds at which the subscriber output file is closed and a new output file opened. When configured, a timestamp is appended to all output filenames for when the file was opened. This parameter does not apply to socket connectors with fstype other than **hdat**.
`rate`	When latency mode is enabled, transmit at this specified rate to generated output files.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`unbufferedoutputstreams`	Specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.
`hdatsecure`	Specifies to use SSH to connect to secure Hadoop plugins. The default value is false. Applies only to the hdat connector.
`hdatsecuresshloginname`	When hdatsecure=true, specifies the SSH login name. Invalid whenhdatsecuresshidentityfile is not configured. The default value is the current user. Applies only to the hdat connector.
`hdatsecuresshidentityfile`	When hdatsecure=true, specifies the SSH identity file. Invalid when hdatsecuresshloginname is not configured. The default value is none. Applies only to the hdat connector.
`hdatsecurehadoopcmdpath`	When hdatsecure=true, specifies the full Hadoop command path on the remote name and data nodes. The default value is "/opt/hadoop/bin/ hadoop". Applies only to the hdat connector.

Table 5-18 Optional Parameters for Publisher File and Socket Connectors

Parameter	Description
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
`addcsvopcode`	Prepends an opcode and comma to incoming CSV events. The opcode is Insert unless publishwithupsert is enabled.
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.
`cefsyslogprefix`	When fstype=cef, specifies that CEF events contain the syslog prefix.

Parameter	Description
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`growinginputfile`	Enables reading from a growing input file by publishers. When enabled, the publisher reads indefinitely from the input file until the connector is stopped or the server drops the connection. This parameter does not apply to socket connectors.
`header`	Specifies the number of input lines to skip before starting publish operations. The default value is 0. This parameter applies only to csv and syslog publish connectors.
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`maxevents`	Specifies the maximum number of events to publish.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window.
`prebuffer`	Controls whether event blocks are buffered to an event block vector before doing any injects. The default value is "false". Not valid with growinginputfile or for a socket connector.
`publishwithupsert`	Build events with opcode=Upsert instead of opcode=Insert.
`rampupsecs`	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with the rate parameter. The default value is 0.
`rate`	Specifies the requested transmit rate in events per second.
`repeatcount`	Specifies the number of times to repeat the publish operation. Applies to publishers that publish a finite number of events. The default value is 0.
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`stripnullsfromcsv`	Strip all nulls from input CSV events.

Common Event Format (CEF) File and Socket Connector

Using this mode publishes data in ArcSight Common Event Format (CEF) to a Source window. By default it processes CEF events that do not contain the syslog prefix. Configure the parameter cefsyslogprefix when events contain that prefix.

The Source window schema must start with the following fields:

```
index*:int32,version:int32,devicevendor:string,deviceproduct:string,
deviceversion:s tring,signatureid:string,name:string,severity:string
```

However, when cefsyslogprefix is configured, the schema must start like this:

```
index*:int32,datetime:date,hostname:string,version:int32,devicevendo
r:string,device
product:string,deviceversion:string,signatureid:string,name:string,s
everity:string
```

The index is added by the connector and serves as the event key field. The rest of the fields are the syslog prefix and required CEF header fields.

The remainder of the schema can include additional fields from the CEF extension, but they are not required. When you include them, the schema field name must match a key in a CEF key-value pair. The schema field type must match the value type in the key-value pair. When there is no key match, the field is set to null. This permits injecting of CEF events with different extension fields into a single Source window.

When a CEF key name contains any characters that are not alphanumeric or underscore, you must replace them with an underscore in the corresponding Source window schema field name.

To properly parse the date in the syslog prefix, configure the dateformat parameter accordingly. For example:

```
"Jan 18 11:07:53" set dateformat to "%b %d %H:%M:%S”
```

CSV File and Socket Connector Data Format

A CSV publisher converts each line into an event. The first two values of each line are expected to be an opcode flag and an event flag. Use the addcsvopcode and addcsvflags parameters when any line in the CSV file does not include an opcode and event flag. When the lines in the CVS file do not include a column with a unique value that can be used as a key, you must specify true for both the autogen-key and noautogenfield connector properties.

HDAT Subscribe Socket Connector Notes

This connector writes Objective Analysis Package Data (HDAT) files in SASHDAT format. This socket connector connects to the name node specified by the fsname parameter. The default CAS or LASR name node port is 15452. You can configure this port. Refer to the SAS Hadoop plug-in property for the appropriate port to which to connect.

The SAS Hadoop plug-in must be configured to permit puts from the service. Configure the property com.sas.cas.hadoop.service.allow.put=true. By default, these puts are enabled. Ensure that this property is configured on data nodes in addition to the name node.

If running multi-threaded (as specified by the hdatnumthreads parameter), the connector opens socket connections to all the data nodes that are returned by the name node. It then writes subscriber event data as Objective Analysis Package Data (HDAT) file parts to all data nodes using the block size specified by the hdfsblocksize parameter. These file parts are then concatenated to a single file when the name node is instructed to do so by the connector.

The connector automatically concatenates file parts when stopped. It might also stop periodically as specified by the optional periodicity and maxfilesize parameters.

By default, socket connections to name nodes and to data nodes have a default time out of five minutes. This time out causes the server to disconnect the edge stream processing connector

after five minutes of inactivity on the socket. It is recommended that you disable the time out by configuring a value of 0 on the SAS Hadoop plug-in configuration property com.sas.cas.hadoop.socket.timeout.

Because SASHDAT format consists of static row data, the connector ignores Delete events. It treats Update events the same as Insert events.

When you specify the hdatlasrhostport or hdatcashostport parameter, the HDAT file is written and then the file is loaded into a LASR or CAS in-memory table. When you specify the periodicity or maxfilesize parameters, each write of the HDAT file is immediately followed by a load of that file.

SAS Hadoop plugins released with and after SAS 9.4M6 require authenticated connections by default. The connector fails to read or write socket connections to these plugins, and logs the following error:

```
"Unknown command: 1 at
com.sas.cas.hadoop.NameNodeService.handleCommand(NameNodeService.jav
a:246)".
```

The connector also supports connecting to secure name and data nodes through SSH. You can enable this mode by configuring the connector hdatsecure parameter. In this mode, SSH channels are opened using the credentials of the current user to run the remote name and data node services using the path /opt/hadoop/bin/hadoop on the remote machines. To use alternate SSH credentials, configure the connector hdatsecuresshloginname and hdatsecuresshidentityfile parameters. To specify an alternate path to the remote name and data node services, configure the connector hdatsecurehadoopcmdpath parameter.

JSON File and Socket Connector Data Format

A JSON publisher requires that the input JSON text conform to the following format:

```
[[event_block_n],[event_block_n+1],[event_block_n+2],....]
```

The outermost brackets define an array of event blocks. Each nested pair of brackets defines an array of events that corresponds to an event block. See “Publish/Subscribe API Support for JSON Messaging” for the semantics to convert this array to and from an event block.

A JSON subscriber writes JSON text in the same format expected by the JSON publisher. In addition to the schema fields, the subscriber always includes an opcode field whose value is the event opcode. Each event block in the output JSON is separated by a comma and new line.

Syslog File and Socket Connector Notes

The syslog file and socket connector are supported only for publisher operations. It collects syslog events, converts them into streaming event blocks, and injects them into one or more Source windows in a running Edge Streaming Analytics model.

The input syslog events are read from a file or named pipe written by the syslog daemon. Syslog events filtered out by the daemon are not seen by the connector. When reading from a named pipe, the following conditions must be met:

- The connector must be running in the same process space as the daemon.
- The pipe must exist.
- The daemon must be configured to write to the named pipe.

You can specify the growinginputfile parameter to read data from the file or named pipe as it is written.

The connector reads text data one line at a time, and parses the line into fields using spaces as delimiters.

The first field in the window schema must be a key field of type INT32. The other fields in the corresponding window schema must be of type ESP_UTF8STR or ESP_DATETIME. The number of schema fields must not exceed the number of fields parsed in the syslog file.

A sample schema is as follows:

```
<schema>
<fields>
<field name='ID' type='int32' key='true'/>
<field name='udate' type='date'/>
<field name='hostname' type='string'/>
<field name='message' type='string'/>
</fields>
</schema>
```

XML File and Socket Connector Data Format

The following XML elements are valid:

- <data>
- <events>
- <transaction>
- <event>
- <opcode>
- <flags>

In addition, any elements that correspond to event data field names are valid when contained within an event. Required elements are <events> and <event>, where <events> contains one or more <event> elements. This defines an event block. Events included within a <transaction> element are included in an event block with type = transactional. Otherwise, events are included in event blocks with type = normal.

A single <data> element always wraps the complete set of event blocks in output XML. A closing <data> element on input XML denotes the end of streamed event blocks. Events are created from input XML with opcode=INSERT and flags=NORMAL, unless otherwise denoted by an <opcode> or <flags> element.

Valid data for the opcode element in input data includes the following:

opcode **Tag Value**	**Opcode**
i	Insert
u	Update
p	Upsert

d	Delete
s	Safe delete

Valid data for the flags element in input data includes the following:

- n — the event is normal
- p — the event is partial update

The subscriber writes only Insert, Update, and Delete opcodes to the output data.

Non-key fields in the event are not required in input data, and their value = NULL if missing, or if the fields contain no data. The subscriber always writes all event data fields.

Any event field can include the partialupdate XML attribute. If its value is true, and the event contains a flags element that specifies partial update, the event is built with the partial update flag. The field is flagged as a partial update with a null value.

5.5.2 Using the File and Socket Adapter

Overview

The file and socket adapter supports publish and subscribe operations on files or socket connections that stream the following data types:

- dfESP binary
- CSV
- XML
- JSON
- syslog (publisher only)
- HDAT (subscriber only)
- CEF (ArcSight Common Event Format) (publisher only)

dfesp_fs_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotation marks (for example, key=\"str,ing\").

Subscriber Usage

Table 5-19 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`fsname=output`	Specifies the subscriber output file or socket "host:port". Leave *host* blank to implement a server.
`fstype=binary | csv | xml | json | hdat | cef`	Specifies the file system type.

Table 5-20 Optional Keys

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`rate=transmitrate`	Specifies the rate to write files when latency mode is enabled.
`rampupsecs=seconds`	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with rate. The default is 0.
`gdconfig=gdconfigfile`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=logconfigfile`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`latencymode=true | false`	When true, specifies latency mode.
`restartonerror= true | false`	When true, restarts the adapter if a fatal error is reported.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`periodicity=seconds`	Specifies the output file time (in seconds). The active file is closed and a new active file opened with the timestamp appended to the filename
`maxfilesize=bytes`	Specifies the output file volume (in bytes). The active file is closed and a new active file opened with the timestamp appended to the filename.
`aggrsize=statsforblocks`	Specifies, in latency mode, statistics for aggregation block size. You can follow the value with a comma-separated value to specify the number of beginning and ending aggregation blocks to ignore in latency calculations.
`hdatfilename=filename`	Specifies the filename to write to Hadoop Distributed File System (HDFS).
`hdatmaxdatanodes=numnodes`	Specifies the maximum number of data nodes. The default value is the number of live data nodes.
`hdfsblocksize=mbytes`	Specifies the Hadoop Distributed File System (HDFS) block size in MB.
`hdfsnumreplicas=number`	Specifies the Hadoop Distributed File System (HDFS) number of replicas. The default value is 1.
`hdatnumthreads=numthreads`	Specifies the adapter thread pool size. A value <= 1 means that no data nodes are used.
`hdatmaxstringlength=size`	Specifies the fixed size to use for string fields in HDAT records. The value must be a multiple of 8.
`hdatlasrhostport=hostport`	Specifies the LASR server host and port.
`hdatlasrkey=path`	Specifies the path to tklasrkey.sh. By default, this file is located in$DFESP_HOME/bin
`hdatcashostport=hostport`	Specifies the CAS server host and port.
`hdatcasusername=username`	Specifies the CAS server user name.
`hdatcaspassword=password`	Specifies the CAS server password.
`latencyblksize=numevents`	Specifies the block size (in number of events) of memory to allocate for storing timestamps when in latency mode. Additional blocks are allocated as required. Required when you specify latencymode.
`unbufferedoutputstreams=true | false`	When true, specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.
`header=headerrow`	For a CSV subscriber, writes a header row that shows comma- separated field names. Set to "full" to include "opcode, flags," in header line.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`hdatsecure=true | false`	When true, specifies to use SSH to connect to secure Hadoop plugins.

Key-Value Pair	Description
`hdatsecuresshloginname=name`	When hdatsecure=true, specifies the SSH login name. Invalid ifhdatsecuresshidentityfile is not configured.
`hdatsecuresshidentityfile=file`	When hdatsecure=true, specifies the SSH identity file. Invalid ifhdatsecuresshloginname is not configured.
`hdatsecurehadoopcmdpath=path`	When hdatsecure=true, specifies the full Hadoop command path on the remote name and data nodes. The default value is "/opt/hadoop/bin/hadoop".

Publisher Usage

Table 5-21 Required Keys

Key-Value Pairs	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`fsname=output`	Specifies the publisher input file or socket "host:port". Leave *host* blank to implement a server.
`fstype=binary | csv | xml | json | hdat | cef`	Specifies the file system type.

Table 5-22 Optional Keys

Key-Value Pairs	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`rate=transmitrate`	Specifies the rate to write files when latency mode is enabled.
`rampupsecs=seconds`	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with rate. The default is 0.
`gdconfig=gdconfigfile`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=logconfigfile`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.

Key-Value Pairs	Description
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`latencymode=true | false`	When true, specifies latency mode.
`restartonerror= true | false`	When true, restarts the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note**: No transport configuration file is required for native transport.
`blocksize=size`	Sets the block size. The default value is 1.
`maxevents=number`	Specifies the maximum number of events to publish.
`transactional=true | false`	When true, specifies that events are transactional.
`growinginputfile=true | false`	When true, read from the growing file
`prebuffer=true | false`	When true, buffer all event blocks prior to publishing.
`header=numlines`	For CSV publisher, the number of header lines to ignore.
`quiesceproject=true | false`	When true, quiesce project after all events are injected into the Source window.
`ignorecsvparseerrors=true | false`	When true, drop event when a field fails CSV parsing and continue.
`csvfielddelimiter=character`	Specifies the character that delimits CSV fields, default = ,
`noautogenfield=true | false`	When true, input events are missing the key field autogenerated by the Source window.
`addcsvopcode=true | false`	When true, prepend opcode to input CSV events.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to insert into input CSV event.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of insert.
`cefsyslogprefix=true | false`	When true, CEF events contain the syslog prefix. By default, they do not.
`repeatcount=number`	Specifies number of times to repeat, default = 0.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

5.6 Using the Event Stream Processor Adapter

The event stream processor adapter enables you to subscribe to a window and publish what it passes into another Source window. This operation can occur within a single event stream processor. More likely it occurs across two event stream processors that are run on different machines on the network.

No corresponding event stream processor connector is provided.

dfesp_esp_adapter –p url -s url <-E tokenlocation>

Parameter	Description
-p url	Specifies the publish standard URL in the form dfESP:// host:port/project/ contquery/window
-s url	Specifies the subscribe standard URL in the form "`dfESP://host:port/ project/contquery/window?snapshot=true |false` When ?snapshot=true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append the following if needed: `?collapse=true | false`"
-E tokenlocation	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.

The eventblock size and transactional nature is inherited from the subscribed window.

For information about implementing hot failover for publisher adapters, see “Publisher Adapter Failover with Kafka (Page 84)”.

5.7 Using the Project Publish Connector

Use the project publish connector to subscribe to event blocks that are produced by a window from a different project within the edge streaming analytics model. The connector receives that window’s event blocks and injects them into its associated window. Window schemas must be identical. When the source project is stopped, the flow of event blocks to the publisher is stopped.

The project publish connector has a reference-counted copy of the events so that only a single copy of the event is in memory.

Table 5-23 Required Parameters for the Project Publish Connector

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`srcproject`	Specifies the name of the source project.
`srccontinuousquery`	Specifies the name of the source continuous query.
`srcwindow`	Specifies the name of the Source window.

Table 5-24 Optional Parameters for the Project Publish Connector

Parameter	Description
`maxevents`	Specifies the maximum number of events to publish.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Database and Big Data Protocols for Cloud Streaming Server

6

6.1 Teradata Connector and Adapter

6.1.1 Using the Teradata Connector

Overview

Teradata Parallel Transporter (TPT) is an object-oriented client application that provides scalable, high-speed, parallel data extraction, loading, and updating. TPT jobs are run using discrete, object-oriented modules that perform these processes across various data stores.

Using the Teradata Connector

The Teradata connector uses the Teradata Parallel Transporter (TPT) API to support subscribe operations against a Teradata server.

The Teradata Tools and Utilities (TTU) package must be installed on the platform running the connector. The run-time environment must define the path to the Teradata client libraries in the TTU. For Teradata ODBC support while using the load operator, you must also define the path to the Teradata TeraGSS libraries in the TTU. To support logging of Teradata messages, you must define the NLSPATH environment variable appropriately. For example, with a TTU installation in /opt/teradata, define NLSPATH as /opt/teradata/ client/version/tbuild/msg64/%N. The connector has been certified using TTU version 14.10.

For debugging, you can install optional PC-based Teradata client tools to access the target database table on the Teradata server. These tools include the GUI SQL workbench (Teradata SQL Assistant) and the DBA/Admin tool (Teradata Administrator), which both use ODBC.

You must use one of three TPT operators to write data to a table on the server:stream, update, or load.

Operator	Description
stream	Works like a standard Edge Streaming Analytics database subscriber connector, but with improved throughput gained through using the TPT. Supports insert, update, and delete events. As it receives events from the subscribed window, it writes them to the target table. If you set the required tdatainsertonly configuration parameter to false, serialization is automatically enabled in the TPT to maintain correct ordering of row data over multiple sessions.
update	Supports insert/update/delete events but writes them to the target table in batch mode. The batch period is a required connector configuration parameter. At the cost of higher latency, this operator provides better throughput with longer batch periods (for example minutes instead of seconds).
load	Supports insert events. Requires an empty target table. Provides the most optimized throughput. Staggers data through a pair of intermediate staging tables. These table names and connectivity parameters are additional required connector configuration parameters. In addition, the write from a staging table to the ultimate target table uses the generic ODBC driver used by the database connector. Thus, the associated connect string configuration and odbc.ini file specification is required. The staging tables are automatically created by the connector. If the staging tables and related error and log tables already exist when the connector starts, it automatically drops them at start-up.

If required, you can configure the tdatauserpwd parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. If you have installed the Edge Streaming Analytics System Encryption and Authentication Overlay, you can use the included OpenSSL executable. Use the following command on the console to invoke OpenSSL to display your encrypted password:

```
echo "tdatauserpwd" | openssl enc -e -aes-256-cbc -a -salt
-pass  pass:"espTDATAconnectorUsedByUser=tdatausername"
```

Then copy the encrypted password into your tdatauserpwd parameter and enable the `tdatauserpwdencrypted parameter.`

Table 6-1 Required Parameters for Teradata Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`desttablename`	Specifies the target table name.
`tdatausername`	Specifies the user name for the user account on the target Teradata server.
`tdatauserpwd`	Specifies the user password for the user account on the target Teradata server.
`tdatatdpid`	Specifies the target Teradata server name

Parameter	Description
`tdatamaxsessions`	Specifies the maximum number of sessions created by the TPT to the Teradata server.
`tdataminsessions`	Specifies the minimum number of sessions created by the TPT to the Teradata server.
`tdatadriver`	Specifies the operator: stream, update, or load.
`tdatainsertonly`	Specifies whether events in the subscriber edge stream processing window are insert only. Must be true when using the load operator.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 6-2 Optional Parameters for Teradata Connectors

Parameter	Description
`rmretdel`	Removes all delete events from event blocks received by the subscriber that were introduced by a window retention policy.
`tdatabatchperiod`	Specifies the batch period in seconds. Required when using the update operator, otherwise ignored.
`stage1tablename`	Specifies the first staging table. Required when using the load operator, otherwise ignored.
`stage2tablename`	Specifies the second staging table. Required when using the load operator, otherwise ignored.
`connectstring`	Specifies the connect string used to access the target and staging tables. See "Using the Database Connector" for Teradata connectstring requirements.
`tdatatracelevel`	Specifies the trace level for Teradata messages written to the trace file in the current working directory. The trace file is named operator1.txt. Default is 1 (TD_OFF). Other valid values are: 2 (TD_OPER), 3 (TD_OPER_CLI), 4 (TD_OPER_OPCOMMON), 5 (TD_OPER_SPECIAL), 6 (TD_OPER_ALL), 7 (TD_GENERAL), and 8 (TD_ROW).
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`tdatauserpwdencrypted`	Specifies that tdatauserpwd is encrypted.

6.1.2 Using the Teradata Subscriber Adapter

The Teradata adapter supports subscribe operations against a Teradata server, using the Teradata Parallel Transporter for improved performance. You must install the Teradata Tools and Utilities (TTU) to use the adapter.

Note

A separate database adapter uses generic ODBC drivers for the Teradata platform.

```
dfesp_tdata_adapter -C key1=val1,key2=val2,key3=val3,…
```

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 6-3 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false. Append?collapse=true | false or ?rmretdel=true | false or both if needed.
`tdatadriver=stream | update | load`	Specifies the Teradata operator. For more information about these operators, see the documentation provided with the Teradata Tools and Utilities.
`tdatausername=username`	Specifies the Teradata user name.
`tdatauserpwd=password`	Specifies the Teradata user password.
`desttablename=name`	Specifies the name of the target table on the Teradata server.
`tdatatdpid=name`	Specifies the name of the target Teradata server.
`tdatamaxsessions=number`	Specifies the maximum number of sessions on the Teradata server. A Teradata session is a logical connection between an application and Teradata Database. It allows an application to send requests to and receive responses from Teradata Database. A session is established when Teradata Database accepts the user name and password of a user.
`tdataminsessions=number`	Specifies the minimum number of sessions on the Teradata server.
`tdatainsertonly=true | false`	When true, specifies that the subscribed window contains insert- only data.

Table 6-4 Optional Keys

Key-Value Pair	Description
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`tdatabatchperiod=seconds`	Specifies the batch period in seconds. This parameter is required when tdatadriver=update.
`stage1tablename=name`	Specifies the name of the first staging table on the Teradata server. This parameter is required when tdatadriver=load.
`stage2tablename=name`	Specifies the name of the second staging table on the Teradata server. This parameter is required when tdatadriver=load.
`connectstring=string`	Specifies the connect string to be used by the ODBC driver to access the staging and target tables on the Teradata server. The string takes the form "DSN=dsn[;uid=uid][;pwd=pwd]". This parameter is required when tdatadriver=load.
`tdatatracelevel=level`	Specifies the trace level for Teradata messages written to the trace file in the current working directory. The trace file is named operator1.txt. Default value is 1 (TD_OFF). Other valid values are: 2 (TD_OPER), 3 (TD_OPER_CLI), 4 (TD_OPER_OPCOMMON), 5 (TD_OPER_SPECIAL), 6 (TD_OPER_ALL), 7 (TD_GENERAL), and 8 (TD_ROW).
`tdatauserpwdencrypted=true | false`	When true, tdatauserpwd is encrypted.

Key-Value Pair	Description
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note**: No transport configuration file is required for native transport.

6.2 Teradata Listener Connector and Adapter

6.2.1 Using the Teradata Listener Connector

Overview

Teradata Listener is an intelligent, self-service solution for ingesting and distributing extremely fast moving data streams in near real-time. It monitors incoming data streams, gathers metadata, and produces metrics for data flow over time. For more information, see the documentation (https://docs.teradata.com/reader/YWVpEBxiUVdaD9qbWzmFdQ/3Y6Oe37c5eYFjGtQC4fQWQ).

Note

Use the Teradata Listener Connector and Adapter with Teradata Listener release 2.02 or higher.

Using the Teradata Listener Connector

The Teradata Listener connector sends data from Edge Streaming Analytics to the Teradata Listener application. Teradata Listener ingests high-volume, real-time data streams and persists the data from those streams to Teradata, Aster, or Hadoop. Listener withstands service interruptions in target systems by buffering data until the target systems become available.

Note

- Because the Teradata Listener connector uses Transport Layer Security (TLS) to establish an encrypted connection to the Teradata Listener Ingest service, the run-time environment must define the path to your TLS libraries (for example, specifying LD_LIBRARY_PATH on Linux platforms).
- Blocks of events are sent to the Teradata Listener Ingest service using a REST API. The Teradata Listener connector supports Listener Ingest REST API (version 1).
- After data is delivered to the Listener Ingest service, Listener queues the data and sends it to all targets that subscribe to that Listener source stream. Targets can be one or more Teradata, Aster, or Hadoop systems, as well as other applications that read from the same Listener source using the Listener Stream service.
- The Teradata Listener connector collects events from Edge Streaming Analytics and transforms them into JSON or plaintext format, as specified by the contentType parameter. The connection to the Listener Ingest service is defined by the ingestUrl and SSLCert parameters. The Listener source (data stream) where the data is delivered is identified by the sourceKey parameter.
- The Teradata Listener connector is a subscribe-only. To send data from Teradata Listener to Edge Streaming Analytics, use the WebSocket connector (Page 190) with contentType=json.

Table 6-5 Required Parameters for Teradata Listener Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`ingestUrl`	Specifies the URL for the Listener Ingest REST API (version 1). This URL must end with **messages**.
`SSLCert`	Specifies the path to a file that contains TLS certificates securely connect to the Listener Ingest service. Listener uses TLS 1.2.
`sourceKey`	Specifies the Listener source secret key that identifies the Listener source feed to which Edge Streaming Analytics sends data. Contact your Teradata Listener administrator to obtain the source secret key.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.
`tdlistenervers ion`	Specifies the Teradata Listener version (for example, 2.02).

Table 6-6 Optional Parameters for Teradata Listener Connectors

Parameter	Description
`ingestBlocksize`	Specifies the maximum number of data rows to send in one Listener Ingest message. Matching the connector block size to the Source window block size is recommended. The default block size is 256.
`contentType`	Specifies the format of the data sent from Edge Streaming Analytics to Listener, either json or plaintext (comma-delimited). The default is "json".
`ingestDelim`	Specifies the character that delimits data rows in a multi-row message from Edge Streaming Analytics to the Listener Ingest REST API. The delimiter must not be a JSON punctuation character. The default is a tilde (~).
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

See also

Edge Streaming Analytics: Security. (http://support.sas.com/documentation/cdlutility/cdlredirect.htm?locale=en&alias=espsec&pubcode=72453&id=titlepage)

6.2.2 Using the Teradata Listener Adapter

The Teradata Listener subscriber adapter provides the same functionality as the Teradata Listener connector (Page 131). It also provides several optional, adapter-only parameters.

dfesp_tdlistener_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotation marks (for example, key=\"str,ing\").

Table 6-7 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`ingestUrl=url`	Specifies the URL for the Listener Ingest REST API (version 1). This URL must end with **messages**.

Key-Value Pair	Description
`SSLCert=path`	Specifies the path to a file that contains TLS certificates for securely connecting to the Listener Ingest service. Listener uses TLS 1.2.
`sourceKey=key`	Specifies the Listener source secret key that identifies the Listener source feed where Edge Streaming Analytics sends data. Contact your Teradata Listener administrator to obtain the source key.

Table 6-8 Optional Keys

Key-Value Pair	Description
`ingestBlocksize=number`	Specifies the maximum number of data rows to send in one Listener Ingest message. The default is 256.
`ingestDelim=character`	Specifies the character that delimits data rows in a multi-row message from Edge Streaming Analytics to the Listener Ingest REST API. The delimiter must not be a JSON punctuation character. The default is tilde (~).
`contentType=json | plaintext`	Specifies the format of the data sent from Edge Streaming Analytics to the Listener. The default is "json".
`debug=true | false`	Enables debugging. The default is false.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Key-Value Pair	Description
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`tdlistenerversion=version`	Specifies the Teradata Listener version (for example, 2.02).

6.3 Using the Cassandra Adapter

Overview

The Cassandra adapter is engineered to handle data to and from Apache Cassandra databases (https://cassandra.apache.org/). The adapter is available in dfx-esp-cassandra-adapter.jar, which bundles the Java publisher and subscriber Edge Streaming Analytics clients. The subscriber client receives Edge Streaming Analytics Engine event blocks and converts the enclosed events to Insert/Update/Delete operations on the target Cassandra keyspace and table. The publisher client converts rows in a Cassandra result set to events and injects them to the source window of an edge streaming engine

The adapter requires the DataStax Java Driver for Apache Cassandra, which you must download from https:// github.com/datastax/java-driver. The adapter was tested with version 3.0.0 of this drive, which supports Cassandra version 1.2 through 3.0 and later. It negotiates the version of native protocol to use during a connection. Other versions of the driver might work with the adapter, but they are not tested. The client target platform must define the environment variable DFESP_DATASTAX_JARS to specify the location of the DataStax Java driver JAR files and other required JAR files. The additional JAR files that you need depend on the version of the Java driver that you download.

The adapter is functionally similar to the Database publisher and subscriber.

Subscriber Usage

For the subscriber, every subscribed event block is separated into separate lists of Inserts and Updates and Deletes. Each list is converted to a set of prepared statements that are added to a set of Insert, Update, and Delete batches. The batches are then executed in the following order:

1. Insert batch
2. Update batch
3. Delete batch

By default, the batches are built and executed once for every subscribed event block. You can configure the

batchsize and batchsecs parameters to modify batching behavior.

dfesp_cassandra_subscriber -C key1=val1,key2=val2,key3=val3,…

Table 6-9 Required Keys

Key-Value Pair	Description
`node=clusternode`	Specify the Cassandra cluster node IP address or host name.
`keyspace=space`	Specify the Cassandra destination keyspace.
`table=table`	Specifies the Cassandra destination table.
`url=url`	Specifies the edge stream processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*?snapshot=true | false<? collapse=true | false>".

Table 6-10 Optional Keys

Key-Value Pair	Description
`loadbalancingpolicy= policy`	Specifies a Cassandra load balancing policy, with optional comma- separated wrapped policies. The default value is "TokenAware,DCAwareRoundRobin".
`batchsize=number`	Specifies the total number of statements per Insert, Update, and Delete set of batches. The default is the event block size.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData %\Mdsp\esa\EdgeStreamingServer\default\javaadapters.config** (Windows) to parse for configuration parameters.
`gdconfigfile=filename`	Specifies the Guaranteed Delivery configuration file for this publisher.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`compression=type`	Specifies Cassandra transport compression, valid values are "NONE", "LZ4", "SNAPPY", default is "NONE".
`username=name`	Specifies the user name to log on to the Cassandra host.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is warning.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`password=password`	Specifies the password to use with the specified user name.
`ssl=true | false`	When true, enables Transport Layer Security (TLS) on the Cassandra connection, using the default platform JSSE options.
`batchsecs=seconds`	Specifies the maximum number of seconds to hold an incomplete batch. The default is 0.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported.

Key-Value Pair	Description
`port=port`	Specifies the Cassandra cluster node port. The default value is 9042.
`retrypolicy=policy`	Specifies a Cassandra retry policy, with optional comma-separated wrapped policies, default is "Default".
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`unloggedbatches= true | false`	Specifies whether Cassandra batch statement are unlogged. The default value is **false**.

Publisher Usage

For the publisher, the user-configured select statement is executed on the target Cassandra keyspace and table. Each row in the result set is converted to an Insert event (or Upsert if so configured), which is injected to the Source window. After these operations, the adapter quits.

dfesp_cassandra_publisher -C *key1*=*val1* , *key2*=*val2*, *key3*=*val3*, …

Table 6-11 Required Keys

Key-Value Pair	Description
`selectstatement=statement`	Specifies the CQL statement to use to pull rows from the Cassandra table.
`node=clusternode`	Specifies the Cassandra cluster node IP address or host name.
`keyspace=space`	Specifies the Cassandra keyspace.
`table=table`	Specifies the Cassandra destination table.
`url=url`	Specifies the edge stream processing standard URL in the following format: "dfESP://*host*:*port*/*project*/*contquery*/*window*".

Table 6-12 Optional Keys

Key-Value Pair	Description
`loadbalancingpolicy= policy`	Specifies a Cassandra load balancing policy, with optional comma- separated wrapped policies. The default value is "TokenAware,DCAwareRoundRobin".
`blocksize=number`	Specifies the number of events per event block. The default value is 1.

Key-Value Pair	Description
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or %ProgramData **%\Mdsp\esa\EdgeStreamingServer \default \javaadapters.config** (Windows) to parse for configuration parameters.
`gdconfigfile=filename`	Specifies the guaranteed delivery configuration file for the client.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`compression=value`	Specifies Cassandra transport compression, valid values are "NONE", "LZ4", "SNAPPY", default is "NONE".
`username=name`	Specifies the user name to log on to the Cassandra host.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is warning.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`password=password`	Specifies the password to use with the specified user name.
`quiesceproject=true|false`	When true, quiesces the project after all events are injected into the Source window.
`transactional=true|false`	When true, specifies that event blocks are transactional. The default event block type is normal.
`ssl=true|false`	When true, enables TLS on the Cassandra connection, using the default platform JSSE options.
`publishwithupsert=true|false`	When true, builds events with opcode=Upsert instead of Insert.
`restartonerror=true|false`	When true, restarts the adapter if a fatal error is reported.
`maxevents=number`	Specifies the maximum number of events to publish.
`port=number`	Specifies the Cassandra cluster node port. The default value is 9042.
`retrypolicy=policy`	Specifies a Cassandra retry policy, with optional comma-separated wrapped policies, default is "Default".
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

6.4 Using the SAS Cloud Analytics Services Adapter

Overview

SAS Cloud Analytic Services (CAS) is a server that is suitable for both on-premises and cloud deployments. The server provides the run-time environment for data management and analytics. The server can run on a single machine or as a distributed server on multiple machines. For more information, see SAS Cloud Analytic Services: Fundamentals (http://support.sas.com/documentation/cdlutility/cdlredirect.htm?locale=en&alias=casfun&pubcode=70935&id=titlepage).

The CAS subscriber adapter enables you to append edge streaming analytics events to a global table in the SAS Cloud Analytic Services server. The table includes a column named "_opcode" that contains the opcode associated with each Edge Streaming Analytics event.

The CAS publisher adapter enables you to fetch all rows from a global table in the SAS Cloud Analytic Services server. The publisher adapter injects those rows as events with opcode=Insert (unless -u is configured) into a

Source window.

The SAS Cloud Analytic Services server requires that client connections be secured with TLS encryption. A Java truststore contains TLS certificates in Java. You must define the DFESP_JAVA_TRUSTSTORE and SSLCALISTLOC environment variables for the client target platform. DFESP_JAVA_TRUSTSTORE sets the location of the .jks file in your truststore. SSLCALISTLOC specifies the location of the .pem file that includes public certificate(s) for trusted certificate authorities (CA).

For more information about SSLCALISTLOC, see Encryption in SAS (http://support.sas.com/documentation/cdlutility/cdlredirect.htm?locale=en&alias=secref&pubcode=69831&id=titlepage).

You must specify caspassword in non-encrypted form when you set the pwdencrypted key to true. The encrypted version of the password can be generated using OpenSSL. The OpenSSL executable is included in the Edge Streaming Analytics System Encryption and Authentication Overlay installation. Use the following command on the console to use OpenSSL to display your encrypted password:

```
echo "caspassword" | openssl enc -e -aes-256–cbc -a -salt -pass
pass:"espCASadapterUsedByUser=casusername"
```

Use the encrypted password to specify caspassword for the caspassword key and enable encryption with the pwdencrypted key.

CAS adapters use dual authentication when connecting to a SAS Cloud Analytic Services server. You need to set up this authentication properly on the SAS Cloud Analytic Services server.

Subscriber Usage

dfesp_cas_adapter -C key1=val1,key2=val2,key3=val3,…

Note

For a subscriber, the specified table in SAS Cloud Analytic Services does not need to exist. When it does, its columns should match a subset of the columns in the event stream processor window. Rows are appended to the existing table. When you do not specify the cassessionid key, the table is promoted to a global table.

Note

By default, the subscriber keeps the addTable action open indefinitely. To periodically close the action in order to avoid having the CAS server time out during periods of inactivity, configure the maxnumrows or periodicity keys.

Table 6-13 Required Keys

Key-Value Pairs	Description
`url=url`	Specifies the edge stream processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*?snapshot=true | false<? collapse=true | false><?rmretdel=true | false>". **Note:** Multiple URLs, separated by commas, are permitted.
`cashostport=url`	Specifies the CAS server host name and port in the form "*host:port*"
`type=sub`	Specifies a subscriber adapter.
`table=table`	Specifies the full name of the CAS table.

Table 6-14 Optional Keys

Key-Value Pairs	Description
`commitblocks=number`	Specifies the number of event blocks to commit to the SAS Cloud Analytic Services table at one time. The default value is 8. This parameter trades throughput for latency. Increase the value to increase the number of event blocks appended to the SAS Cloud Analytic Services table before being committed.
`buffersize=size`	Specifies the buffer size. The default value is 512. This buffering parameter specifies the number of table rows that the adapter buffers before sending them to the SAS Cloud Analytic Services.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default \javaadapters.config** (Windows) to parse for configuration parameters.
`pwdencrypted= true | false`	When true, the CAS password is encrypted. By default, it is not.

Key-Value Pairs	Description
`gdconfigfile=filename`	Specifies the guaranteed delivery configuration file for the client.
`cassessionid=id`	Specifies the UUID of the session to which to connect. If not specified, the SAS Cloud Analytic Services adapter creates a new SAS Cloud Analytic Services session.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default value is native.
`loglevel=severe | warning | info`	Specifies the application logging level. The default level is warning.
`caslib=libname`	Specifies the SAS Cloud Analytic Services library containing the SAS Cloud Analytic Services table. The default is the currently active SAS Cloud Analytic Services library.
`casusername=username`	Specifies the username used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casoauthtoken* is not specified.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`casoauthtoken=token`	Specifies the OAuth token that is used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casusername* is not specified.
`caspassword=password`	Specifies the password used to authenticate the SAS Cloud Analytic Services session. When *caspassword* is not specified and *casusername* is specified, the password is extracted from that user's **authoinfo/netrc** file.
`columns=subset`	Specifies a subset of columns to read or write to and from the SAS Cloud Analytic Services table. Use the form c1_name,...cn_name. **Note**: When the column names in the SAS Cloud Analytic Services table match all the field names in the edge stream processing window, you do not need to use this key. The edge stream processing schema corresponds one–to–one with the SAS Cloud Analytic Services schema (except for the additional reserved key field required for a publisher and the opcode field written by a subscriber).
`restartonerror= true | false`	When true, restarts the adapter when a fatal error is reported.javaadapters.config parameter name: restartonerror
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note**: No transport configuration file is required for native transport.

Key-Value Pairs	Description
`maxnumrows=number`	Specifies the maximum number of table rows per addTable action. By default, the action is kept open indefinitely.
`periodicity=number`	Specifies the maximum number of seconds to keep the addTable action open. By default, the action is kept open indefinitely.

Publisher Usage

dfesp_cas_adapter -C key1=val1, key2=val2, key3=val3, …

Note

For a publisher, the specified table in the SAS Cloud Analytic Services must exist. Its columns should match a subset of the columns in the event stream processor Source window. In addition, when you do not specify the cassessionid parameter, the table must be global. The first field in the event stream processor Source window schema is reserved for the row number and must be defined as an INT64. This first field serves as the event's key field.

Table 6-15 Required Keys

Key-Value Pair	Description
`url=url`	Specifies the edge streaming processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*".
`cashostport=url`	Specifies the CAS server host name and port in the form "*host:port*"
`type=pub`	Specifies a publisher adapter.
`castable=table`	Specifies the full name of the CAS table.

Table 6-16 Optional Keys

Key-Value Pair	Description
`blocksize=size`	Specifies the number of events to be flushed to the server. The default value is 1.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData** **%\Mdsp\esa\EdgeStreamingServer** **\default\javaadapters.config** (Windows) to parse for configuration parameters.
`quiesceproject=true | false`	When true, specifies to quiesce the project after all events are injected into the Source window. **javaadapters.config** parameter name: quiesceproject
`pwdencrypted= true | false`	When true, the CAS password is encrypted. By default, it is not.

Key-Value Pair	Description
`transactional= true | false`	When true, event blocks are transactional. By default, they are normal.
`gdconfigfile=filename`	Specifies the guaranteed delivery configuration file for the client.
`cassessionid=id`	Specifies the UUID of the session to which to connect. If not specified, the SAS Cloud Analytic Services adapter creates a new SAS Cloud Analytic Services session.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default value is native.
`loglevel=severe | warning | info`	Specifies the application logging level. The default level is warning.
`caslib=libname`	Specifies the SAS Cloud Analytic Services library containing the SAS Cloud Analytic Services table. The default is the currently active SAS Cloud Analytic Services library.
`casusername=username`	Specifies the username used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casoauthtoken* is not specified.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`casoauthtoken=token`	Specifies the OAuth token that is used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casusername* is not specified.
`caspassword=password`	Specifies the password used to authenticate the SAS Cloud Analytic Services session. When caspassword is not specified and casusername is specified, the password is extracted from that user's authoinfo/netrc file.
`columns=subset`	Specifies a subset of columns to read or write to and from the SAS Cloud Analytic Services table. Use the form c1_name,...cn_name. **Note**: When the column names in the SAS Cloud Analytic Services table match all the field names in the edge stream processing window, you do not need to use this key. The edge stream processing schema corresponds one–to–one with the SAS Cloud Analytic Services schema (except for the additional reserved key field required for a publisher and the opcode field written by a subscriber).
`publishwithupsert=true | false`	When true, builds events with opcode=Upsert. By default, events are built with opcode=Insert. `javaadapters.config parameter name:publishwithupsert`
`restartonerror= true | false`	When true, restarts the adapter when a fatal error is reported. `javaadapters.config parameter name: restartonerror`

Key-Value Pair	Description
`maxevents=number`	Specifies the maximum number of events to publish.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note**: No transport configuration file is required for native transport.

Note

When the SAS Cloud Analytic Services table contains fewer columns than the edge stream processing window, use columns to choose the subset of edge stream processing fields to read or write to or from the table. For a publisher, unmatched fields in the window are loaded with null values. For a subscriber, unmatched fields in the window are left out of the variable list that is appended to the SAS Cloud Analytic Services table.

Data Type Mappings

The SAS Cloud Analytic Services to SAS Event Steam Processing data type mappings are as follows:

SAS Cloud Analytic Services type	Edge Streaming Analytics data type
INT32	INT32
INT64	INT64
DOUBLE	DOUBLE
DATE	DATETIME
CHAR	UTF8STR or RUTF8STR
VARCHAR	UTF8STR or RUTF8STR
DECSEXT	MONEY
TIME	INT64
DATETIME	TIMESTAMP
VARBINARY	BINARY

Note

The SAS Cloud Analytic Services type DECQUAD and BINARY do not have an Event Stream Processor data type mapping.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

See also

Dual Authentication (http://support.sas.com/documentation/cdlutility/cdlredirect.htm?)

6.5 Using the HDFS (Hadoop Distributed File System) Adapter

The HDFS adapter resides in dfx-esp-hdfs-adapter.jar, which bundles the Java publisher and subscriber Edge Streaming Analytics clients. The subscriber client receives event blocks and writes events in CSV format to an HDFS file. The publisher client reads events in CSV format from an HDFS file and injects event blocks into a Source window of an engine.

The target HDFS and the name of the file within the file system are both passed as required parameters to the adapter.

The subscriber client enables you to specify values for HDFS block size and number of replicas. You can configure the subscriber client to periodically write the HDFS file using the optional periodicity or maxfilesize parameters. If so configured, a timestamp is appended to the filename of each written file.

You can configure the publisher client to read from a growing file. In that case, the publisher runs indefinitely and publishes event blocks whenever the HDFS file size increases.

You must define the DFESP_HDFS_JARS environment variable for the client target platform. This variable specifies the locations of the Hadoop JAR files and your core-site.xml file.

List the JAR files in this order: hadoop-common-*.jar , hadoop-hdfs-*.jar , common hadoop JARs. For example, in Linux you would run the following from the command line:

```
$ export DFESP_HDFS_JARS=/usr/local/hadoop-2.5.0/share/hadoop/
common/hadoop-common-2.5.0.jar:
/usr/local/hadoop-2.5.0/share/hadoop/hdfs/hadoop-hdfs-2.5.0.jar:
/usr/local/hadoop-2.5.0/share/hadoop/common/lib/*:/usr/local/
hadoop-2.5.0/conf
```

Subscriber usage:

dfesp_hdfs_subscriber

-C key1=val1,key2=val2,key3=val3,…

Table 6-17 Required Keys

Key-Value Pair	Description
`hdfs=target`	Specifies the target file system, in the form "hdfs:// host:port". Specifies the file system that is normally configured in property fs.defaultFS in core-site.xml.
`inputfile=file`	Specifies the input CSV file, in the form "/path/ filename.csv".
`url=url`	Specifies the edge stream processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*".

Table 6-18 Optional Keys

Key-Value Pair	Description
`hdfsblocksize=size`	Specifies the HDFS block size in MB. The default is 64MB.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData** **%\Mdsp\esa\EdgeStreamingServer\default\javaadapters.config** (Windows) to parse for configuration parameters.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`gdconfigfile=file`	Specifies the guaranteed delivery configuration file for the client.
`jaasconf=file`	Specifies the location of the JAAS configuration on the local file system. This file is used for Kerberos authentication to the Hadoop grid. By default, there is no authentication.
`krb5conf=file`	Specifies the location of the Kerberos 5 configuration file on the local file system. This file is required for Kerberos authentication to the Hadoop grid. The default value is **/opt/mdsp/ esa/ config/etc/EdgeStreamingServer/ default/krb5.conf** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default\krb5.conf** (Windows).
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default value is native.
`maxfilesize=size`	Specifies the output file periodicity in bytes.
`hdfsnumreplicas=number`	Specifies the HDFS number of replicas. The default value is 1.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is warning.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`periodicity=seconds`	Specifies the output file periodicity in seconds. Invalid if data is not insert-only.
`opaquestringfield=field`	Specifies the subscribed window field from which to extract the output line.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note**: No transport configuration file is required for native transport.

Publisher usage:

dfesp_hdfs_publisher -C key1=val1, key2=val2, key3=val3, …

Table 6-19 Required Keys

Key-Value Pair	Description
`hdfs=target`	Specifies the target file system, in the form "hdfs:// host:port". Specifies the file system that is normally configured in property fs.defaultFS in core-site.xml.
`inputfile=file`	Specifies the input CSV file, in the form "/path/ filename.csv".
`url=url`	Specifies the edge stream processing standard URL in the following format: "dfESP://*host.port*/*project*/*contquery*/*window*".

Table 6-20 Optional Keys

Key-Value Pair	Description
`addcsvopcode=true | false`	When true, prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is true. **javaadapters.config** parameter name: addcsvopcode
`addcsvflags=type`	Specifies the event type in insert into input CSV events. Valid values are normal and partialupdate.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`quiesceproject=true | false`	When true, quiesce the project after all events are injected into the Source window.
`restartonerror=true | false`	When true, restarts the adapter when a fatal error is reported. **javaadapters.config** parameter name: restartonerror
`maxevents=number`	Specifies the maximum number of events to publish.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note**: No transport configuration file is required for native transport.
`blocksize=size`	Specifies the number of events per event block. The default value is 1.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\javaadapters.config** (Windows) to parse for configuration parameters.

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`ignorecsvparseerrors=true | false`	When true, drops an event when a field in an input CSV event cannot be parsed. An error is logged and publishing continues. `javaadapters.config parameter name:ignorecsvparseerrors`
`gdconfigfile=file`	Specifies the guaranteed delivery configuration file.
`jaasconf=file`	Specifies the location of the JAAS configuration on the local file system. This file is used for Kerberos authentication to the Hadoop grid. By default, there is no authentication.
`krb5conf=file`	Specifies the location of the Kerberos 5 configuration file on the local file system. This file is required for Kerberos authentication to the Hadoop grid. The default value is **/opt/mdsp/esa/ config/etc/EdgeStreamingServer/ default/krb5.conf** (Linux) or %**ProgramData%\Mdsp\esa\EdgeStreamingServer\default\krb5.conf** (Windows).
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default is native.
`csvfielddelimiter=delimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, input events are missing the key field that is autogenerated by the Source window. By default, events contain all schema fields.
`publishwithupsert=true | false`	When true, build events with opcode=Upsert. By default, events are built with opcode=Insert. `javaadapters.config parameter name:publishwithupsert`
`transactional=true | false`	When true, event blocks are transactional. By default, they are normal.
`stripnullsfromcsv=true | false`	When true, strip all null characters from input CSV events.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

Messaging systems for Cloud Streaming Server 7

7.1 Using the Twitter Publisher Adapter

Before you use the Twitter publisher adapter, review the terms of service available at https://twitter.com/en/tos. This adapter works with the Standard and Premium levels of service.

The Twitter publisher adapter consumes Twitter streams and injects event blocks into Source windows of an engine. The adapter provides the following capabilities:

- Subscribes to Twitter streams and receives a continuous flow of tweets as soon as they are tweeted.
- Filters the incoming streams using the following standard features provided by Twitter:
  - Follows specific users.
  - Tracks specific keywords.
  - Filters specific geographical locations.
  - Filters specific languages.
  - Feceives random generated tweets.
  - Receives only RT tweets. This requires specific Twitter account authorization.
  - Receives only tweets that include links. This requires specific Twitter account authorization.
  - Receives full “fire hose.” This requires specific Twitter account authorization.
- Publishes received tweets to a Source window for processing inside a model.

Usage:

dfesp_twitter_publisher -C key1=val1,key2=val2,key3=val3,…

Table 7-1 Required Keys

Key-Value Pairs	Description
`method=sample | filter | links | firehose | retweeet`	Specifies the access method used to listen to Twitter streams. For a description of each access method, refer to the table that follows. The default method is sample.
`url=url`	Specifies the edge streaming processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*".

Table 7-2 Twitter Stream Access Methods

Access Method	Limitations
sample	Starts listening on random sample of all public statuses. The default access level provides a small proportion of the firehose. The "Gardenhose" access level provides a proportion more suitable for data mining and research applications that need a statistically significant sample.
filter	Starts consuming public statuses that match one or more filter predicates. At least one predicate parameter, follow, locations, or track must be specified. Multiple parameters can be specified to enable most clients to use a single connection to the Streaming API. Placing long parameters in the URL can cause the request to be rejected for excessive URL length. The default access level allows up to 200 track keywords, 400 follower user IDs and 10 1-degree location boxes. Increased access levels enable the following: • 80,000 follower user IDs ("shadow" role) • 400,000 follower user IDs ("birddog" role) • 10,000 track keywords ("restricted track" role) • 200,000 track keywords ("partner track" role) • 200 10-degree location boxes ("locRestricted" role) Increased track access levels also pass a higher proportion of statuses before limiting the stream. For more information about specifying parameters, see https://developer.twitter.com/en/docs.
firehose	Starts listening on all public statuses. Available only to Twitter approved parties and requires a signed agreement to access.
retweet	Starts listening on all retweets. The retweet stream is not a generally available resource. Few applications require this level of access. Creative use of a combination of other resources and various access levels can satisfy nearly every application use case.
link	Starts listening on all public statuses containing links. Available only to Twitter approved parties and requires a signed agreement to access.

Table 7-3 Optional Keys

Key-Value Pair	Description
`languagelist=list`	Specifies a comma-separated list of BCP 47 language identifiers to return only Tweets that have been detected as being written in the specified languages. For example, connecting with "en,fr" streams only Tweets detected to be in the English or French language. **Note**: This parameter must be used in conjunction with at least one of the filter predicates of the filter access method. The -alanguagelist parameter is supported only with the filter access method, specified at the command line with the -mfilter argument. The filter access method requires at least one of its supported filter predicates. For more information, see Table 8.508.
`blocksize=number`	Specifies the number of events per event block.
`prevcount=number`	Used with filter, link, or retweet method. Indicates the number of previous statuses to stream before transitioning to the live stream. The default value is 0. The supplied value can be an integer from 1 to 150000 or from -1 to -150000. When a positive number is specified, the stream transitions to live values after the backfill has been delivered to the client. When a negative number is specified, the stream disconnects after the backfill has been delivered to the client, which might be useful for debugging. This parameter requires elevated access to use. See Streaming API request parameters (https://dev.twitter.com/streaming/overview/request-parameters) for more information.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData %\Mdsp\esa\EdgeStreamingServer\default\javaadapters.config** (Windows) to parse for configuration parameters.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`dumpfile=file`	Specifies the full path of a file in which to dump binary event blocks for replay. Use dfESP_fs_adapter to replay event blocks.
`followlist=list`	Specifies the list of user IDs to follow with the filter method in the form "*user1,user2,user3*". The default access level allows up to 400 follower user IDs. See Streaming API request parameters (https://dev.twitter.com/streaming/overview/request-parameters) for more details.
`gdconfigfile=file`	Specifies the guaranteed delivery configuration file for the client.

Key-Value Pair	Description
`tracklist=list`	Specifies the list of keywords to track with the filter method in the form "*keyword1,keyword2,keyword3*". The default access level allows up to 200 track keywords.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default value is native.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is warning.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`locationlist=list`	Specifies the list of locations to track with the filter method in the form of a comma-separated list of *longitude,latitude* pairs. Each pair specifies a set of bounding boxes with the south-west corner of the bounding box coming first. For example: "-122.75,36.8,-121.75,37.8" for San Francisco. Note that bounding boxes do not act as filters for other filter parameters. The default access level allows up to 10 1-degree location boxes.
`publishwithupsert=true | false`	When true, builds events with opcode=Upsert instead of Insert.
`transactional=true | false`	When true, specifies that event blocks are transactional. The default event block type is normal.
`twitterpropfile=profile`	Specifies the name of the file in your configuration directory to parse for Twitter configuration parameters. Default is **twitter-publisher.properties**.
`restartonerror=true | false`	When true, restart the adapter after a fatal error.
`maxevents=number`	Specifies the maximum number of events to publish.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note:** No transport configuration file is required for native transport.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka".

Before you run the Twitter adapter, download the twitter4j-core and twitter4j-stream JAR files from twitter4j.org and copy them to $DFESP_HOME/lib. Versions 4.0.2 and 4.0.4 are certified.

To authorize a Twitter user account to be used by the Twitter adapter, grant access to your Twitter user account on behalf of the SAS Twitter app ID. The Twitter properties file located in the Edge Streaming Analytics configuration directory must include an encrypted access token in order for a user account to access the adapter. By default, the Twitter properties file is named twitterpublisher.properties and does not include the access token. To generate and write an encrypted access token to the properties file, run the dfesp_twitter_auth utility that is located in $DFESP_HOME/bin. Once the token is included in the properties file, verify that the adapter

runs successfully. You do not need to run the dfesp_twitter_auth utility more than once for the same Twitter account.

If you need to add other settings such as proxy servers, API stream URLs, or trace settings, see http:// twitter4j.org/en/configuration.html#Streaming and edit your Twitter properties file accordingly.

The Source window of the model that the Twitter adapter publishes to must have the following schema:

```
tw_ID*:int64,tw_AuthorScreenName:string,tw_AuthorId:int64,tw_AuthorD
escrip
tion:string,tw_AuthorFavouritesCount:int32,tw_AuthorFollowersCount:i
nt32,tw

_AuthorFriendsCount:int32,tw_AuthorProfileImageURL:string,tw_AuthorL
ang:str
ing,tw_AuthorLocation:string,tw_AuthorName:string,tw_AuthorTimeZone:
string,
tw_AuthorURL:string,tw_AuthorStatusesCount:int32,tw_AuthorListedCoun
t:int32

,tw_Text:string,tw_CreatedAt:stamp,tw_CurrentUserRetweetId:int64,tw_
Favorit
eCount:int32,tw_GeolocLatitude:double,tw_GeolocLongitude:double,tw_I
nReplyT
oScreenName:string,tw_InReplyToStatusId:int64,tw_InReplyToUserId:int
64,tw_L
ang:string,tw_RetweetCount:int32,tw_RetweetedStatusId:int64,tw_Sourc
eText:s
tring,tw_Source:string,tw_isFavorited:int32,tw_isPossiblySensitive:i
nt32,tw

_isRetweet:int32,tw_isRetweeted:int32,tw_isTruncated:int32,tw_PlaceF
ullName

:string,tw_MentionedUserNames:string,tw_MentionedUserScreenNames:str
ing,tw_
MentionedUserIds:string,tw_StatusLinks:string,tw_StatusTags:string,f
_method

:string,f_followed_IDs:string,f_tracked_terms:string,f_GeoLocations:
string, f_Languages:string
```

Table 7-4 Schema Fields for the Source Window

Field	Type	Description
tw_ID	int64	The ID of the status
tw_AuthorScreenName	string	The screen name of the user ID associated with the status
tw_AuthorId	int64	The user ID associated with the status.
tw_AuthorDescription	string	The description of the author
tw_AuthorFavouritesCount	int32	The number of favorites of the author

Field	Type	Description
tw_AuthorFollowersCount	int32	The number of followers of the author
tw_AuthorFriendsCount	int32	The number of friends of the author
tw_AuthorProfileImageURL	string	The profile image url of the author
tw_AuthorLang	string	The preferred language of the author
tw_AuthorLocation	string	The location of the author
tw_AuthorName	string	The name of the author
tw_AuthorTimeZone	string	The time zone of the author
tw_AuthorURL	string	The URL of the author
tw_AuthorStatusesCount	int32	The status count of the author
tw_AuthorListedCount	int32	The number of public lists the author is listed on, or -1 if the count is unavailable.
tw_Text	string	The text of the Status
tw_CreatedAt		The date of creation of the Status
tw_CurrentUserRetweetId	int64	The authenticating user's ID of this tweet, or -1L when the tweet was created before this feature was enabled.
tw_FavoriteCount	int32	The number of times this Tweet has been "favorited" by Twitter users.
tw_GeolocLatitude	double	The location that this tweet refers to, if available (can be null).
tw_GeolocLongitude	double	The location that this tweet refers to, if available (can be null).
tw_InReplyToScreenName	string	The screen name of the status, this status is replying to.
tw_InReplyToStatusId	int64	The ID of the status, this status is replying to.
tw_InReplyToUserId	int64	The ID of the user, this status is replying to.
tw_Lang	string	The two-letter ISO language code if the status if available.
tw_RetweetCount	int32	The number of times this tweet has been retweeted, or -1 when the tweet was created before this feature was enabled. Please note that as tweets are received, in real time, this value is always 0, except when the prevcount parameter is specified.
tw_RetweetedStatusId	int64	The ID of the status, this status is a retweet of.
tw_SourceText	string	The source this status has been emitted from, in the form of the source HTML tag.
tw_Source	string	The source device or application this status has been emitted from.
tw_isFavorited	string	True when the status is flag as a Favorite.

Field	Type	Description
tw_isPossiblySensitive	int32	True when the status contains a link that is identified as sensitive.
tw_isRetweet	int32	True when the status is retweet.
tw_isRetweeted	int32	True when the status is retweeted.
tw_isTruncated	int32	True when the status has been truncated.
tw_PlaceFullName	string	The place attached to this status.
tw_MentionedUserNames	string	A list of names of user mentioned in the tweet.
tw_MentionedUserScreenNames	string	A list of screen names of user mentioned in the tweet.
tw_MentionedUserIds	string	A list of IDs of user mentioned in the tweet.
tw_StatusLinks	string	A list of links mentioned in the tweet.
tw_StatusTags	string	The list of mentioned hashtags without #
f_method	string	The method specified on the followlist parameter of the adapter used to query Twitter streams (source, filter, retweet, firehose, links).
f_followed_IDs	string	The list of user IDs specified on the followlistparameter of the adapter to filter Twitter streams.
f_tracked_terms	string	The list of keywords specified on the tracklistparameter of the adapter to filter Twitter streams.
f_GeoLocations	string	The list of geolocations specified on the locations parameter of the adapter to filter Twitter streams.
f_Languages	string	The list of languages specified on the languagesparameter of the adapter to filter Twitter streams.

You can add the following optional fields to your Source window in order to grab media entities that are present in tweets:

```
tw_media_display_url:string
tw_media_expanded_url:string
tw_media_id:string
tw_media_media_url:string
tw_media_source_status_id:string
tw_media_type:string
tw_media_url:string
```

When you include the tw_media_source_status_id fields, tweets are converted to JSON and parsed for that field, because there is no supporting API call. For this to work, Twitter4J (http://twitter4j.org/en/) requires that jsonStoreEnabled=true be configured in the twitterpublisher.properties file. A run-time exception is thrown if that property is not configured, and the following message is logged:

```
Failed to parse tweet raw json: java.lang.IllegalStateException:
Apparently jsonStoreEnabled is not set to true. You may need to add
this line to twitterpublisher.properties: jsonStoreEnabled=true
```

Because a media entity is an array, each tweet that contains a media entity could contain multiple values of these fields. In this case, these fields contain comma-separated values that are in the same order as in the media entity array.

If a tweet contains no media entity, then the source window fields are left at null.

When a media entity is present in a tweet but not all fields are present in all array positions, the corresponding Source window field shows nothing between commas.

7.2 Using the Twitter Gnip Publisher Adapter

7.2.1 Using the Adapter

Before you use the Twitter Gnip publisher adapter, review the terms of service available at https://twitter.com/en/ tos. This adapter works with the Enterprise level of service.

The Twitter Gnip adapter is a Python script that invokes the Edge Streaming Analytics C publish/subscribe and JSON libraries. The adapter builds event blocks from a Twitter Gnip firehose stream and publishes them to a source window. Access to this Twitter stream is restricted to Twitter-approved parties. Access requires a signed agreement.

The adapter taps into the Gnip stream through a keep-alive connection to the configured URL. The connection uses the configured user ID and password credentials. Every subsequent JSON response is converted to an event block that uses the JSON parsing rules described in "Publish/Subscribe API Support for JSON Messaging" on page 837. The corresponding Source window must contain fields that correspond to everyreceived JSON key unless you specify the adapter parameter esp-ignoremissingschemafields. Eventsare built with the Insert opcode unless you configure the adapter to use Upsert instead. If needed, a JSON filtering function can be enabled using the esp-matchsubstrings parameter.

You assign key field(s) in the Source window. If it is not practical to use a JSON key as a key field, one or both of the following key fields can be added to the Source window schema:

```
eventindex*:int64
```

```
adapterindex*:string
```

The eventindex key field is incremented for every event that is built from input JSON. The adapterindex key field contains a constant GUID value that is unique for every instance of the adapter. The combination of these two key fields enables multiple instances of the adapter to publish events to the same Source window without duplicating keys.

Usage:

dfesp_twittergnip_publisher -esp-url url -streampassword streampassword -streamurl streamurl -streamuserid streamuserid < <-esp-dateformat dateformat > > <-esp-ignoremissingschemafields > <-esp-logconfigfile logconfigfile > < <-esp-loglevel loglevel > > <-

esp-matchsubstrings substrings > <-esp-publishwithupsert> <- gnipkeepalive gnipkeepalive><--maxrecords maxrecords >

Table 7-5 Parameters for the Twitter Gnip Publisher Adapter

Parameter	Description
`-esp-dateformat dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`-esp-ignoremissingschemafields`	Specifies to ignore and continue when the JSON key is missing in schema. By default, an error is reported, and the process quits when the key is missing.
`-esp-logconfigfile logconfigfile`	Specifies the name of the log configuration file.
`-esp-loglevel loglevel`	Specifies the logging level. Permitted values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL. The default value is INFO.
`-esp-matchsubstrings substrings`	Specifies a list of comma–separated fieldname:value pairs, where input events that do not contain a value substring in the fieldname string field are dropped. The comparison is case-insensitive.
`-esp-publishwithupsert`	Publish events with opcode=Upsert. By default, events are published with opcode=Insert.
`-esp-url url`	Specifies the publisher standard URL in the form "dfESP:// host:port/project/ continuousquery/window"
`-gnipkeepalive gnipkeepalive`	Specifies the keep-alive value on Gnip connections. The default is 30 seconds.
`-maxrecords maxrecords`	Sets the maximum number of records to process. By default, there is no maximum.
`-streampassword streampassword`	Specifies the Gnip password.
`-streamurl streamurl`	Specifies the Gnip streaming URL, in the form https:// stream.gnip.com:443/accounts/account/ publishers/twitter/ streams/stream/ label.json
`-streamuserid streamuserid`	Specifies the Gnip username.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

7.2.2 Using the Twitter Gnip Query Registration Utility

There is a companion Python script in **$DFESP_HOME/lib** called GNIPQueryMgr.py. Run the script with no parameters to see usage.

This script enables you to view, add, and remove specific query strings registered with Gnip (that is, the Gnip rules that are associated with a specific Gnip stream and label).

Viewing registered queries shows all matching queries unless you use the groupname parameter to filter them. This parameter corresponds to the tag key that is associated with the

value key in a rules array entry. A rule is shown when the specified groupname appears as a substring in the rule's tag value.

Adding or removing a query requires a corresponding groupname parameter, in addition to the specified query string.

7.3 Using the BoardReader Publisher Adapters

The BoardReader application is a feed aggregator. The site boardreader.com enables you to search message boards, websites, blogs, and other social media.

The BoardReader adapters provided by Edge Streaming Analytics are a collection of Python scripts. These scripts invoke the C publish/subscribe and JSON libraries to build event blocks from a BoardReader feed and then publish them to a source window. Each script provides access to a specific BoardReader content source. These content sources include message boards, blogs, news, YouTube, and reviews.

The parameters required to configure access to the content source are listed in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/boardreader-*-config.txt** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\boardreader-*- config.txt** (Windows). Sample versions of these files are included in your Edge Streaming Analytics distribution.

You must configure the following parameters:

- customer_project_id
- customer_query_id
- key
- query

You can leave the other parameters at their default values as shown in the sample files.

The adapters run configured queries in independent threads and wait for those threads to complete. As each thread receives responses, it publishes event blocks to the edge stream processing engine from the same thread. Each query gathers responses from the content source filtered per that query's filter_date_from and filter_date_to parameters.

When the adapter is in streaming mode (which is the default setting of the request_mode parameter), it waits for all threads to complete. It then repeats the process after an interval specified by the request_interval and request_interval_unit parameters. Before running the next iteration of queries, each query's filter_date_from and filter_date_to parameters are updated in the configuration file. This ensures that its next invocation continues to stream from the content source uninterrupted.

Every received JSON response is converted to an event block using the JSON parsing rules described in "Publish/Subscribe API Support for JSON Messaging" on page 837" . The corresponding Source window must contain fields that correspond to every received JSON key unless you specify the adapter parameter

esp-ignoremissingschemafields. Events are built with the Insert opcode unless you configure the adapter

to use Upsert instead. If needed, a JSON filtering function can be enabled using the esp-matchsubstrings

parameter.

You assign key field(s) in the Source window. If it is not practical to use a JSON key as a key field, one or both of the following key fields can be added to the Source window schema:

```
eventindex*:int64
```

```
adapterindex*:string
```

The eventindex key field is incremented for every event that is built from input JSON. The adapterindex key field contains a constant GUID value that is unique for every instance of the adapter. The combination of these two key fields enables multiple instances of the adapter to publish events to the same Source window without duplicating keys.

The Source window schema must also include the following two fields:

- ProjectID:string
- Query:string

The contents of these fields mirrors the values in the customer_project_id and customer_query_id

configuration parameters, for correlation purposes.

Usage:

dfesp_*_boardreader -config-txt configtxt -esp-url url <-dev-mode-history-to-disk> <-dev-mode-resp-to-disk> <- email-from emailfrom > <-email-level emaillevel > <-email-smtphost emailsmtphost <-email-subject emailsubject

>> <-email-to emailto > <-esp-dateformat dateformat > <-esp-logconfigfile logconfigfile > <-esp-loglevel loglevel> <-esp-ignoremissingschemafields> <-esp-matchsubstrings substrings > <-esp-publishwithupsert>

Table 7-6 Parameters for the BoardReader Publisher Adapters

Parameter	Description
`-config-txt configtxt`	Specifies the name of a text file in the configuration directory that contains BoardReader content source configuration parameters. See the sample files in your configuration directory.
`-dev-mode-history-to-disk`	Provides additional debugging writes SQLite database formatted documents to disk in your configuration directory unless you have configured the work_dir parameter.
`-dev-mode-resp-to-disk`	Provides additional debugging writes responses to disk in your configuration directory unless you have configured the work_dir parameter.
`-email-from emailfrom`	Specifies the source address for email logging messages.
`-email-level emaillevel`	Specifies the email logging level. Permitted values are DEBUG, INFO, WARNING, ERROR, and CRITICAL. The default value is CRITICAL.
`-email-smtphost emailsmtphost`	Specifies the SMTP host for email logging messages.
`-email-subject emailsubject`	Specifies the subject line for email logging messages.
`-email-to emailto`	Specifies the destination address for email logging messages.

Parameter	Description
`-esp-dateformat dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`-esp-ignoremissingschemafields`	Specifies to ignore and continue when the JSON key is missing in schema. By default, an error is reported and the process quits when the key is missing.
`-esp-logconfigfile logconfigfile`	Specifies the name of the log configuration file.
`-esp-loglevel loglevel`	Specifies the logging level. Permitted values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL. The default value is INFO.
`-esp-matchsubstrings substrings`	Specifies a list of comma separated fieldname:value pairs, where input events that do not contain a value substring in the fieldname string field are dropped. The comparison is case-insensitive.
`-esp-publishwithupsert`	Specifies to publish events with opcode = Upsert. By default, events are published with opcode = Insert.
`-esp-url url`	Specifies the publisher standard URL in the form "dfESP://host:port/project/ continuousquery/window"

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

7.4 Using the Java Message Service (JMS) Adapter

The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API to send messages between two or more clients. The Java Message Service (JMS) adapter bundles the Java publisher and subscriber clients. Both are JMS clients. The subscriber client receives event blocks and is a JMS message producer. The publisher client is a JMS message consumer and injects event blocks into a source window of an engine. The adapter resides in dfx-esp-jms-adapter.jar.

The subscriber client requires a command line parameter that defines the type of JMS message used to contain events. The publisher client consumes the following JMS message types:

- BytesMessage - an Event Streams Processing event block
- TextMessage - an Event Streams Processing event in CSV or XML format
- MapMessage - an Edge streaming analytics event with its field names and values mapped to corresponding MapMessage fields

A JMS message in TextMessage format can contain an XML document encoded in a third-party format. You can substitute the corresponding JAR file in the class path in place of dfx-esp-jms-native.jar, or you can use the -x switch in the JMS adapter script. Currently, dfx-esp-jms-axeda.jar is the only supported alternative.

The JMS password must be passed in unencrypted form unless -E is configured. The encrypted version of the password can be generated using OpenSSL, which must be installed on your system. When you install the Edge Streaming Analytics System Encryption and

Authentication Overlay, you install the included OpenSSL executable. Use the following command on the console to use OpenSSL to display your encrypted password:

echo "jmspassword" | openssl enc -e -aes-256–cbc -a -salt -pass

pass:"espJMSadapterUsedByUser=jmsuserid"

Specify the encrypted password for your jmspassword parameter and enable encryption with the -E parameter.

When running with an alternative XML format, you must specify the JAXB JAR files in the environment variable DFESP_JAXB_JARS. You can download JAXB from https:// jaxb.java.net .

The client target platform must connect to a running JMS broker (or JMS server) . The environment variable DFESP_JMS_JARS must specify the location of the JMS broker JAR files. The clients also require a jndi.properties file, which you must specify through the DFESP_JMS_PROPERTIES environment variable. This properties file specifies the connection factory that is needed to contact the broker and create JMS connections, as well as the destination JMS topic or queue.

You can override the default JNDI names that are looked up by the adapter to find the connection factory and queue or topic. Do this by configuring the jndidestname and jndifactname adapter parameters. When the destination requires credentials, you can specify these as optional parameters on the adapter command line.

A sample jndi.properties file is included in your configuration directory .

Subscriber usage:

dfesp_jms_subscriber -C key1=val1,key2=val2,key3=val3,…

Table 7-7 Required Keys

Key-Value Pair	Description
`messagetype="BytesMessage" | "TextMessage" | "MapMessage"`	Specifies the JMS message type.
`url=url`	Specifies the edge stream processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*?snapshot=true | false<? collapse=true | false>".

Table 7-8 Optional Keys

Key-Value Pair	Description
`jndifactname=name`	Specifies the JNDI name to look up for the JMS connection factory. The default value is "ConnectionFactory". This option overrides settings in **jndi.properties**.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default \javaadapters.config** (Windows) to parse for configuration parameters.

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`pwdencrypted=true | false`	When true, the JMS password is encrypted. By default, it is not.
`protofile=file`	Specifies the .proto file that contains the message used for Google Protocol buffer support.
`gdconfigfile=filename`	Specifies the Guaranteed Delivery configuration file.
`jmsuserid=userid`	Specifies the user ID for the JMS destination.
`jndidestname=name`	Specifies the JNDI name to look up for the JMS destination. The default value is "jmsDestination". This option overrides settings in jndi.properties.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default is native.
`csvmsgpereventblock= true | false`	For CSV, specifies to send one message per event block. The default is one message per transactional event block and one message per event in a non-transactional event block. In a multi-event message, CSV lines are delimited by the "line.separator" system property.
`csvmsgperevent= true | false`	For CSV, specifies to send one message per event. The default is one message per transactional event block and one message per event in a non-transactional event block. In a multi-event message, CSV lines are delimited by the "line.separator" system property.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is warning.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`jmspassword=password`	Specifies the password for the JMS destination.
`opaquestringfield=field`	Specifies the subscribed window field from which to extract the JMS TextMessage.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported.
`xmllib="axeda" | "turkcell"`	Specifies the JAR file derived from a third-party .xsd file that defines the XML contained in a JMS TextMessage. By default, the TextMessage contains CSV.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Publisher Usage:

dfesp_jms_publisher -C key1=val1, key2=val2, key3=val3…

Table 7-9 Required Keys

Key-Value Pair	Description
`url=url`	Specifies the edge streaming processing standard URL in the following format: "dfESP://*host:port/project/contquery/window*".

Table 7-10 Optional Keys

Key-Value Pair	Description
`addcsvopcode=true | false`	When true, prepends an opcode and comma to input CSV events, or received messages of type = MapMessage that are missing the"eventOpcode" field. The opcode is Insert unless -s is enabled.
`pwdencrypted=true | false`	When true, the JMS password is encrypted. By default, it is not.
`pwdencrypted="normal" | "partialupdate"`	Specifies the event type to insert into input CSV events (with comma), or received messages of type = MapMessage that are missing the "eventFlags" field.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported.
`maxevents=number`	Specifies the maximum number of events to publish.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default is native.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note**: No transport configuration file is required for native transport.
`jndifactname=name`	Specifies the JNDI name to look up for the JMS connection factory. The default value is "ConnectionFactory". This option overrides settings in **jndi.properties**.
`blocksize=number`	Specifies the number of events per event block. The default value is 1.
`configfilesection=section`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData** **%\Mdsp\esa\EdgeStreamingServer \default\javaadapters.config** (Windows) to parse for configuration parameters.

Key-Value Pair	Description
`dateformat=format`	For messages of type TextMessage, specifies the format of datetime and timestamp fields. Specify a Java SimpleDate-Format pattern in quotation marks.
`ignorecsvparseerrors=true | false`	When true, drop an event and continue whenever a field fails CSV parsing.
`protofile=file`	Specifies the .proto file that contains the message used for Google Protocol buffer support.
`gdconfigfile=filename`	Specifies the Guaranteed Delivery configuration file for this publisher.
`jmsuserid=userid`	Specifies the user ID for the JMS destination.
`jndidestname=name`	Specifies the JNDI name to look up for the JMS destination. The default value is "jmsDestination". This option overrides settings in **jndi.properties**.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default is native.
`csvfielddelimiter=delimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, input events are missing the key field that is autogenerated by the source window. By default, events contain all schema fields.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is warning.
`jmspassword=password`	Specifies the password for the JMS destination.
`opaquestring=true | false`	When true, do not parse TextMessage. Instead, copy complete message to a ESP_UTF8STR field.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter.
`publishwithupsert=true | false`	When true, build events with opcode = upsert. By default, opcode=insert.
`transactional=true | false`	When true, event blocks are transactional. By default, event blocks are normal.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.
`xmlib="axeda" | "turkcell"`	Specifies the JAR file derived from a third-party .xsd file that defines the XML contained in a JMS TextMessage. By default, the TextMessage contains CSV.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

7.5 IBM WebSphere MQ Connector and Adapter

7.5.1 Using the IBM WebSphere MQ Connector and Adapter

Using the IBM WebSphere MQ Connector

The IBM WebSphere MQ connector (MQ) supports the IBM WebSphere Message Queue Interface for publish and subscribe operations. The subscriber receives event blocks and publishes them to an MQ queue. The publisher is an MQ subscriber, which injects received event blocks into Source windows.

The IBM WebSphere MQ Client run-time libraries must be installed on the platform that hosts the running instance of the connector. The run-time environment must define the path to those libraries (for example, specifying **LD_LIBRARY_PATH** on Linux platforms).

Note

To use this connector.

- Locate the reference to this connector in the connectors: excluded: section of the file **esp-properties.yml**.
- Set the value to **false**.

The connector operates as an MQ client. It requires that you define the environment variable **MQSERVER** to specify the connector's MQ connection parameters. This variable specifies the server's channel, transport type, and host name. For more information, see your WebSphere documentation.

An MQ connector can read and write messages to and from an MQ queue or topic, depending on the settings of the mqqueue and mqtopic parameters. In addition, an MQ subscriber requires a parameter that defines the message format used to publish events to MQ. An MQ publisher can consume any message type that is produced by an MQ subscriber.

Specifically, when the message payload is text instead of binary, the format name in the message header must be equal to MQSTR. Any other value causes the publisher to assume that the payload is binary.

Alternatively, you can configure the ignoremqmdformat parameter to ignore the message format parameter. In this case, the publisher connector assumes the format is compatible with the mqtype parameter setting.

An MQ subscriber or publisher reading or writing to or from an MQ queue must have the mqqueue parameter configured. Configuring the mqtopic parameter is not required. If reading from a WebSphere topic, an MQ publisher requires two additional parameters that are related to durable subscriptions. The publisher always subscribes to an MQ topic using a durable subscription. This means that the publisher can re-establish a former subscription and receive messages that had been published to the related topic while the publisher was disconnected.

These parameters are as follows:

- Subscription name, which is user supplied and uniquely identifies the subscription
- The MQ queue opened for input by the publisher

The MQ persistence setting of messages written to MQ by an MQ subscriber is always equal to the persistence setting of the MQ queue.

Both publishers and subscribers support a **usecorrelid** parameter that causes the correlation ID on every MQ message to be copied to or from a correlid field in an Edge Streaming Analytics event. This only applies when the mqtype parameter is **csv** or **json**. The **correlid** field can be anywhere in the window schema but must be of type ESP_UTF8STR. For a subscriber writing multiple events to an MQ message, the correlation ID value is copied from the first event in the event block. For a publisher reading multiple events from an MQ message, the same correlation ID value is written to every event created from the message. Because MQ correlation ID is a byte string (not a character string), the value of the **correlid** field in an Edge Streaming Analytics event is always base64 encoded.

Use the following parameters for MQ connectors.

Table 7-11 Required Parameters for Subscriber MQ Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`mqtype`	Specifies binary, csv, json, xml, protobuf, or the name of a string field in the subscribed window schema.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 7-12 Required Parameters for Publisher MQ Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`mqtype`	Specifies binary, csv, json, xml, protobuf, or opaquestring. For opaquestring, the Source window schema is assumed to be index:int64,message:string.

Table 7-13 Optional Parameters for Subscriber MQ Connectors

Parameter	Description
`mqtopic`	Specifies the MQ topic name. Required if mqqueue is not configured.
`mqqueue`	Specifies the MQ queue name. Required if mqtopic is not configured.
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`queuemanager`	Specifies the MQ queue manager.

Parameter	Description
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when mqtype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when mqtype is not set to **protobuf**.
usecorrelid	Copies the value of the **correlid** field in the event to the MQ message correlation ID.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Table 7-14 Optional Parameters for Publisher MQ Connectors

Parameter	Description
mqtopic	Specifies the MQ topic name. Required if mqqueue is not configured.
mqqueue	Specifies the MQ queue name. Required if mqtopic is not configured.
mqsubname	Specifies the MQ subscription name. Required if mqqueue is not configured.
blocksize	Specifies the number of events to include in a published event block. The default value is 1.
transactional	Sets the event block type to transactional. The default event block type is normal.

Parameter	Description
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`queuemanager`	Specifies the MQ queue manager.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`protofile`	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when mqtype is not set to **protobuf**.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when mqtype is not set to **protobuf**.
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the **,** character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window.
`publishwithupsert`	Builds events with opcode=Upsert instead of Insert.
`addcsvopcode`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are **normal** and **partialupdate**.
`usecorrelid`	Copies the value of the MQ message correlation ID into the **correlid** field in every Edge Streaming Analytics event.
`ignoremqmdformat`	Specifies to ignore the value of the Message Descriptor Format parameter, and assume the message format is compatible with the mqtype parameter setting.
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events.

7.5.2 Using the IBM WebSphere MQ Adapter

Overview

The IBM WebSphere MQ adapter supports publish and subscribe operations on IBM WebSphere Message Queue systems. To use this adapter, you must install IBM WebSphere MQ Client run-time libraries and define the environment variable MQSERVER to specify the adapter's MQ connection parameters.

dfesp_mq_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Subscriber Usage

Table 7-15 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`mqtype=binary | csv | json | xml | protobuf | opaquestring`	Specifies a message form of binary, csv, json, xml, or protobuf. For opaquestring, the Source window schema is assumed to be index:int64,message:string.

Table 7-16 Optional Keys

Key-Value Pair	Description
`mqtopic=name`	Specifies the MQ topic name.
`queuemanager=name`	Specifies the MQ queue manager name.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when mqtype is not set to **protobuf**.

Key-Value Pair	Description
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when mqtype is not set to **protobuf**.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, , orrabbitmq kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/sas/viya/ config/etc/ SASEventStreamProcessingEngine/ default/connec-tors.config** (Linux) or **%ProgramData%** **\SAS\Viya\SASEventStreamProcessingEngine** **\default\connectors.config** (Windows) to parse for configura-tion parameters. Specify the value as [configfilesection].
`usecorrelid=true | false`	When true, copies the MQ correlation ID to or from the **correlid** string field in the Edge Streaming Analytics event. For non-binary formats only.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`mqqueue=name`	Specifies the MQ queue name.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the de-fault is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`csvmsgperevent= true | false`	When true, for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock=true | false`	When true, for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII repre-sentation of a double. The default value is 6.

Publisher Usage

Table 7-17 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`mqtype=binary | csv | json | xml | protobuf | opaquestring`	Specifies a message form of binary, csv, json, xml, or protobuf. For opaquestring, the Source window schema is assumed to be index:int64,message:string.

Table 7-18 Optional Keys

Key-Value Pair	Description
`mqtopic=name`	Specifies the MQ topic name.
`queuemanager=name`	Specifies the MQ queue manager name.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when mqtype is not set to **protobuf**.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when mqtype is not set to **protobuf**.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/sas/viya/ config/etc/ SASEventStreamProcessingEngine/ default/connectors.config** (Linux) or **%ProgramData%** **\SAS\Viya\SASEventStreamProcessingEngine** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Key-Value Pair	Description
`usecorrelid=true | false`	When true, copies the MQ correlation ID to or from the **correlid** string field in the SAS Event Stream Processing event. For non-binary formats only.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`mqqueue=name`	Specifies the MQ queue name.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`mqsubname=name`	Specifies the MQ subscription name. Required if mqqueue is not configured.
`mqsubqueue=name`	Specifies the MQ queue name. **Note:** For backward compatibility, use -Q only.
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=delimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When true, specifies that input events are missing the key field that is autogenerated by the Source window.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`addcsvopcode=true | false`	When true, prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is true.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`ignoremqmdformat=true | false`	Specifies to ignore the value of the Message Descriptor Format parameter, and assume that the message format is compatible with the mqtype parameter setting.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

7.6 Using the SMTP Connector and Adapter

7.6.1 Using the SMTP Subscriber Connector

Overview

Simple Mail Transfer Protocol (SMTP) is an internet standard for electronic mail (email) transmission. Every mail system uses SMTP when sending and receiving mail from outside its own system, even when it uses nonstandard protocols to access accounts on its own mail server.

Using the SMTP Subscribe Connector

You can use the Simple Mail Transfer Protocol (SMTP) subscribe connector to email window event blocks or single events, such as alerts or items of interest. This connector is subscribe-only. The connection to the SMTP server uses port 25. No user authentication is performed, and the protocol runs unencrypted.

The email sender and receiver addresses are required information for the connector. The email subject line contains a standard event stream processor URL in the form dfESP://host:port/project/contquery/ window, followed by a list of the key fields in the event. The email body contains data for one or more events encoded in CSV format.

The parameters for the SMTP connector are as follows:

Table 7-19 Required Parameters for the SMTP Connector

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`smtpserver`	Specifies the SMTP server host name or IP address.
`sourceaddress`	Specifies the email address to be used in the "from" field of the email.
`destaddress`	Specifies the email address to which to send the email message.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 7-20 Optional Parameters for SMTP Connectors

Parameter	Description
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`emailperevent`	Specifies true or false. The default is false. If false, each email body contains a full event block. If true, each mail body contains a single event.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa \EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`connectionpereventblock`	When true, reconnect and disconnect to and from the SMTP server for every event block. When false, maintain a persistent connection to the SMTP server. The default value is false.

7.6.2 Using the SMTP Subscriber Adapter

The SMTP subscriber adapter is subscriber only. Publishing to a Source window is not supported. This adapter forwards subscribed event blocks or single events as email messages to a configured SMTP server, with the event data in the message body encoded in CSV format.

dfesp_smtp_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 7-21 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**. Append ?snapshot=true |false. Append?collapse=true | false or ?rmretdel=true | false or both if needed.
`smtpserver=name`	Specifies the SMTP server name.
`sourceaddress=saddress`	Specifies the source email address.
`destaddress=daddress`	Specifies the destination email address

Table 7-22 Optional Keys

Key-Value Pair	Description
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSet-PubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`emailperevent=true | false`	When true, send an email message for every event instead of one per event block.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Key-Value Pair	Description
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`connectionpereventblock=true | false`	When true, reconnect and disconnect to and from the SMTP server for every event block. When false, maintain a persistent connection to the SMTP server. The default value is false.

7.7 Using the REST Subscriber Adapter

The REST adapter supports subscribe operations to generate HTTP POST requests to a configured REST service. For each subscribed event, the adapter formats a JSON string that uses all fields of the event unless you configure the opaquejson parameter. In that case, only one field is used. After formatting the string, the adapter forwards the JSON through an HTTP POST to the REST service that is configured in the adapter parameter resturl. You can also configure the HTTP Content-Type and the number of retries when an HTTP POST fails. Additional HTTP POST headers can be specified with the httprequestproperties parameter.

The HTTP response can be forwarded to an unrelated Source window. This requires building an event from the JSON formatted response and forwarding it to the Edge Streaming Server URL that is configured in the espresprurl adapter parameter.

Any field names in the subscribed window schema that contain an underscore are converted into a nested JSON field. For example, field name foo_bar produces the following JSON: "foo": {"bar":"value". The field name foobar produces the following JSON: "foobar":"value".

When the JSON response includes one or more arrays, one of the following conditions must be true in order to successfully build events:

- The array is an outer array that contains the entire response. In this case, an event is built for every entry in the array.
- The array contains multiple values for some key. In this case, all other keys at the same nesting level must contain values in an array of the same size. Then events are built for every array index, using values from that index in each array at that nesting level.

The response event fields are appended to the fields in the original subscribed event. Thus, you must be careful to ensure that the beginning fields in the schema of the Source window in espresprurl exactly match the complete schema of the subscribed window. Also, the response fields must follow the beginning fields.

Each HTTP request-response action is run in a separate adapter thread. In this way, adapter memory usage does not grow with queued subscribed events waiting synchronously for the previous request-response to complete.

dfesp_rest_subscriber -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotation marks (for example, key=\"str,ing\").

Table 7-23 Required Keys

Key-Value Pair	Description
`resturl=url`	Specifies the URL of the target REST service.
`httpcontenttype=value`	Specifies the value of the content-type string used in the HTTP post.
`url=url`	Specifies the edge streaming processing standard URL in the following format: "dfESP://*host:port*/*project*/*contquery*/*window*?snapshot=true|false<? collapse=true|false>".

Table 7-24 Optional Keys

Key-Value Pair	Description
`configfilesection=name`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/javaadapters.config** (Linux) or **%ProgramData %\Mdsp\esa\EdgeStreamingServer\default\javaadapters.config** (Windows) to parse for configuration parameters.
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`espresprul=url`	Specifies the publish/subscribe standard URL to which responses should be published.
`gdconfigfile=filename`	Specifies the guaranteed delivery configuration file for the client.
`ignorefailedhttpconnects=true| false`	When true, drop the subscribed event and continue when an HTTP connect fails. The default is true.
`opaquejson=field`	Specifies a subscribed window field from which to extract the complete JSON request. This is in contrast to building the request from all window fields. The field must have type ESP_UTF8STR.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.

Key-Value Pair	Description
`maxnumthreads=num`	Specifies the maximum number of threads used by the adapter. This limits the number of concurrent connections to the REST service. The default value is 32.
`loglevel=severe | warning | info`	Specifies the application logging level.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`httpretries=number`	Specifies the number of times to retry a failed HTTP post. The default value is 0. A value of -1 specifies to retry the post infinitely.
`httprequestproperties=list`	Specifies any number of additional headers to include in the HTTP POST request to the REST service. Specify as semicolon– separated <*key*>:<*value*> pairs. For keys that support multiple values, separate those values with a comma.
`httpstatuscodes=list`	Specifies a comma-separated list of acceptable status codes in response to the HTTP post. The default required response is 201.
`httpretryinterval=seconds`	Specifies the number of seconds between HTTP post retries. The default is 0.
`restartonerror=true|false`	When true, restarts the adapter when a fatal error is reported.
`transportconfigfile=file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the pubsublib parameter: For solace, the default file is **solace.cfg**. For tervela, the default file is **client.config**. For rabbitmq, the default file is **rabbitmq.cfg**. For kafka, the default file is **kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Other protocols for Cloud Streaming Server 8

8.1 Sniffer Connector and Adapter

8.1.1 Using the Sniffer Publish Connector

Oveview

A packet sniffer is a software program or hardware device that can intercept and log traffic that passes over a digital network or part of a network. As data stream over the network, the sniffer captures packets and decodes its raw data.

Using the Sniffer Publish Connector

The sniffer connector captures packets from a local network interface in promiscuous mode and builds an event per received packet to be injected into a source window. An instance of the connector is configured with the interface name, a protocol, and a comma separated list of fields to be extracted and included in the event.

Additional connectors can be instanced to capture additional packets in any combination of interface/protocol/ fields, and injected to any Source window.

Note

To use this connector:

- Locate the reference to this connector in the connectors: excluded: section of the file esp-properties.yml.
- Set the value to false.

Protocol support is currently limited to the following:

- HTTP packets sent to port 80 over TCP. To capture HTTP packets that you send to other ports (for example, 8080), configure the list of ports in thehttpports parameter.
- Radius Accounting-Request packets sent to port 1813 over UDP. Only attribute values between 1 and 190 inclusive are supported. If you capture the Attribute-Specific field of a Vendor-Specific attribute, you must configure the vendorid and vendortype connector parameters.
- TLS Client Hello packets sent to port 443 over TCP or UDP.
- Traffic on other ports is supported only to the extent that the IP source and destination address, TCP or UDP source and destination port, and the verbatim payload are captured. Other packet fields are not available for capture.

Support is included for an optional 802.1Q VLAN tag header following the Ethernet header, but in all other cases the IP header must directly follow the Ethernet header.

The connector uses the libpcap libraries, which are not shipped with ESP. You must install these libraries separately on the target machine. They are available for download at http://www.tcpdump.org or, for Microsoft Windows, http://www.winpcap.org.

Most kernels protect against applications opening raw sockets. On Linux platforms, the connector logs the following error message unless the application is given permission to open raw sockets:

“You don't have permission to capture on that device (socket: Operation not permitted)”

To grant suitable permissions to the server that is running the connector, run the following command: setcap cap_net_raw,cap_net_admin=eip executable. You must have root privileges to run the command.

A side effect of granting these permissions is that the application no longer uses the shell’s LD_LIBRARY_PATH environment variable. For the Edge Streaming Analytics server application, this means that it must have an alternative method of finding shared objects in $DFESP_HOME/lib. Use the ldconfig command to update the shared library cache with the $DFESP_HOME/lib directory before running the Edge Streaming Analytics server.

The packetfields configuration parameter uses Edge Streaming Analytics schema-like syntax, and must match the Source window schema. There are three exceptions:

- The Source window schema must contain an additional index:int64 field that contains an increasing index value and that serves as the key field.
- The Source window schema must also contain an additional frame_time:stamp field that contains the timestamp created by the pcap driver.
- When the optional addtimestamp connector configuration parameter is specified, the Source window schema must also contain a ts:stamp field to hold the timestamp. This field is created by the connector and holds the current time.

The index and frame_time fields must be the first two fields in the window schema. The ts field must be the last field in the window schema.

When the blocksize parameter is configured with a value greater than 1, the connector builds event blocks of size no greater than the configured blocksize. It can build event blocks with a smaller size when the pcap driver buffer has filled up, or when the read timeout on the socket opened by the pcap driver has expired. This ensures that the connector injects all events available from packets received on the interface as soon as possible, without having to wait to fill an event block.

When the protocol parameter is not set to 80, 1813, or 443 and httpports is not configured, the fields supported in packetfields are as follows:

- The IP source and destination address
- The TCP or UDP source and destination ports
- payload:string

When a received packet is malformed or contains an invalid parameter or length, the connector generally logs an info level message, ignores the packet, and continues.

For testing, you can receive packets from a .pcap capture file instead of a network interface. You can specify the name of this capture file in the connector interface parameter. When the connector cannot find a matching interface name, it treats the name as a .pcap filename.

Table 8-1 Required Parameters for the Sniffer Connector

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`interface`	Specifies the name of the network interface on the local machine from which to capture packets.
`protocol`	Specifies the port number associated with the protocol type of packets to be captured. You can specify this as a comma-separated list of port numbers.
`packetfields`	Specifies the packet fields to be extracted from a captured packet and included in the published event. Use Edge Streaming Analytics schema syntax. This value does not include the index:int64 frame_time:stamp, or ts:stamp fields. Separate nested fields with an underscore character. Match field names to standard names as displayed by the Wireshark open-source packet analyzer. You must remove spaces and hyphens to maintain field name integrity. An example for Radius is as follows: radius_AcctStatus-Type:int32,radius_EventTimestamp:sta mp,radius_FramedI-PAddress:string,radius_UserName:stri ng. An example for HTTP is as follows: ip_Source:string,ip_Destination:string,http_Host:stri ng,http_Referer:string,http_User-Agent:string,http_GET _RequestURI:string. An example for TLS is as follows: ssltls_Handshake_ClientHello_GMTUnixTime:date,ssltls_ Handshake_ClientHello_SessionID:string,ssltls_Handsha ke_ClientHello_Cypher-Suites:string,ssltls_Handshake_C lientHello_Compression-Methods:string,ssltls_Handshake _ClientHello_Extensions_ServerName:string. If protocol is not set to 80, 1813, or 443 and httpports is not configured, the only supported values are as follows: • the IP source and destination address • the TCP or UDP source and destination • portspayload:string

Table 8-2 Optional Parameters for the Sniffer Connector

Parameter	Description
`transactional`	Sets the event block type to transactional. The default value is "normal".
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.

Parameter	Description
addtimestamp	Specifies to append an ESP_TIMESTAMP field to each published event. The field value is the current time when the packet was received by the connector. This field must be present in the Source window schema, but it is not required in the connector packetfields parameter.
configfilesection	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
vendorid	Specifies the vendor-Id field to match when capturing the Attribute-Specific field in a Vendor-Specific attribute in a Radius Accounting-Request packet.
vendortype	Specifies the vendor-Type field to match when capturing the Attribute-Specific field in a Vendor-Specific attribute in a Radius Accounting-Request packet.
indexfieldname	Specifies the name to use instead of index for the index:int64 field in the Source window schema.
publishwithupsert	Specifies to build events with opcode=Upsert instead of opcode=Insert.
pcapfilter	Specifies a filter expression as defined in the pcap documentation. Passed to the pcap driver to filter packets received by the connector.
httpports	Specifies a comma-separated list of destination ports. All sniffed packets that contain a specified port are parsed for HTTP GET parameters. The default value is 80.
ignorenopayloadpackets	Specifies whether to ignore packets with no payload, as calculated by subtracting the TCP or UDP header size from the packet size. The default value is "false".
maxevents	Specifies the maximum number of events to publish.

8.1.2 Using the Sniffer Publisher Adapter

The sniffer adapter supports publish operations of events that are created from packets captured from a local network interface in promiscuous mode. You must install the libpcap runtime libraries in order to use this adapter.

dfesp_sniffer_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Table 8-3 Required Keys

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**.
`interface=networkinterface`	Specifies the name of the network interface on the local machine from which to capture packets.
`protocol=portnumber`	Specifies the port number associated with the protocol type of packets to be captured.
`packetfields=list`	Specifies a list of fields to be extracted from captured packets. You must specify "payload:string" when the protocol type is not 80 (http) or 1813 (radius).

Table 8-4 Optional Keys

Key-Value Pair	Description
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C+ + `C_dfESPpubsubSetPubsubLib`() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in /opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`blocksize=size`	Sets the block size. The default value is 1.
`transactional=true | false`	When true, events are transactional.
`addtimestamp= true | false`	When true, appends an ESP_TIMESTAMP field to each published event.
`vendorid=field`	Specifies the vendor ID field to match when capturing the Attribute- Specific field in a Vendor-Specific attribute in a Radius Accounting- Request packet.
`vendortype=field`	Specifies the vendor type field to match when capturing the Attribute- Specific field in a Vendor-Specific attribute in a Radius Accounting- Request packet.
`indexfieldname=name`	Specifies the name to use instead of index for the index:int65 field in the Source window schema.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.

Key-Value Pair	Description
`pcapfilter=expression`	Specifies a filter expression as defined in the pcap documentation. The value is passed to the pcap driver in order to filter packets received by the adapter.
`httpports=list`	Specifies a comma-separated list of destination ports. All sniffed packets that contain a specified port are parsed for HTTP GET parameters. The default value is 80.
`ignorenonpayloadpackets=true | false`	When true, specifies to ignore packets that have no payload, as calculated by subtracting the TCP or UDP header size from the packet size.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

8.2 WebSocket Connector

8.2.1 Using the WebSocket Connector

Overview

The WebSocket Protocol enables two-way communication between a client that is running untrusted code in a controlled environment and a remote host that accepts the communication that results from that code. For more information about the WebSocket protocol, see RFC-6455 (https://tools.ietf.org/html/rfc6455).

The WebSocket connector enables you to read data over a WebSocket-based connection and publish the data into Edge Streaming Analytics. The connector supports XML and JSON over the WebSocket connection and provides a fully functional API that enables you to parse and transform the data into streaming events.

The WebSocket connector uses event loop technology to parse, transform, and publish the data into Edge Streaming Analytics.

For XML and JSON, the connector reads the WebSocket data until it can build an object of the appropriate type. It then uses functions to process the object and publish events.

The WebSocket connector can be used only to publish.

Table 8-5 Required Parameters for WebSocket Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`url`	Specifies the URL for the WebSocket connection.
`configUrl`	Specifies the URL for the connector configuration file. This configuration file contains information about the transformation steps required to publish events. For example, configuration files, see the WebSocket connector examples located in the examples folder of your SAS Event Stream Processing installation directory: (Linux) **$DFESP_HOME/examples/xml/ ws_connector_json/ config.xml or DFESP_HOME/examples/xml/ ws_connector_xml/config.xml** (Windows) **DFESP_HOME\examples\xml\ws_connector_json\config.xml orDFESP_HOME\examples\xml\ws_connector_xml\config.xml**
`contentType`	Specifies XML or JSON as the type of content received over the WebSocket connection.

Table 8-6 Optional Parameters for WebSocket Connectors

Parameter	Description
`sslCertificate`	Specifies the location of the TLS certificate to use when connecting to a secure server.
`sslPassphrase`	Specifies the password for the TLS certificate.
`requestHeaders`	Specifies a comma-separated list of request headers to send to the server. The list must consist of name-value pairs in name:value format.
`maxevents`	Specifies the maximum number of events to publish.

Example: Using the WebSocket Connector to Publish from Teradata Listener

Use the WebSocket connector to publish from sources with subscribe-only SAS Event Stream Processing connectors.

You can send data from the Teradata Listener Stream service to SAS Event Stream Processing using the WebSocket connector. Use cases include using SAS Event Stream Processing to filter, aggregate, score, or detect patterns in a Listener data stream. The transformed data can then be inserted back into another Listener source to be loaded to Teradata, Aster, or Hadoop. The following XML example follows the syntax for declaring a WebSocket connector that is attached to the Teradata Listener Stream service:

```
<connectors>
<connector class='websocket'>
<properties>
<property  name='type'>pub</property>
```

```
<property    name='url'>wss://listener-streamer-services-
poc.labs.teradata.com/v1/streamer/
444186fb-b652-485f-9dc3-c00c60863476?secret=e52378aa-b7aa-4722-a226-
ca18225481f9</property>
<property  name='contentType'>json</property>
<property  name='configUrl'>file://ws.xml</property>
<property   name='sslCertificate'>/etc/pki/tls/certs/listener.crt</
property>
</properties>
</connector>
</connectors>
```

For the url WebSocket connector parameter, specify the URL for the Listener Stream API (version 1) that includes the Stream identifier and the Stream secret key. To obtain these values, contact your Teradata Listener administrator.

For the SSLCert WebSocket connector parameter, specify the path to a file that contains TLS certificates for securely connecting to the Listener Stream service. Listener uses TLS 1.2.

8.3 Tervela Connector and Adapter

8.3.1 Using the Tervela Data Fabric Connector

Overview

Tervela Data Fabric enables you to capture, share, and distribute data from a number of enterprise and cloud data sources. It is designed for high-performance applications such as financial trading and settlements, payment processing, network monitoring, and analytics.

Using the Tervela Data Fabric Connector

The Tervela Data Fabric connector communicates with a software or hardware-based Tervela Data Fabric for publish and subscribe operations.

A Tervela subscriber connector receives events blocks and publishes to the following Tervela topic: “SAS.ENGINES.enginename.projectname.queryname.windowname.OUT.”

A Tervela publisher connector reads event blocks from the following Tervela topic: “SAS.ENGINES.enginename.projectname.queryname.windowname.IN” and injects them into Source windows.

As a result of the bus connectivity provided by the Tervela Data Fabric connector, the engine does not need to manage individual publish/subscribe connections. A high capacity of concurrent publish/subscribe connections to a single Edge Streaming Analytics engine is achieved.

You must install the Tervela run-time libraries on the platform that hosts the running instance of the connector. The run-time environment must define the path to those libraries (specify LD_LIBRARY_PATH on Linux platforms, for example).

Note

To use this connector:

- Locate the reference to this connector in the connectors: excluded: section of the file esp-properties.yml.
- Set the value to false.

The Tervela Data Fabric Connector has the following characteristics:

- It works with binary event blocks. No other event block formats are supported.
- It operates as a Tervela client. All Tervela Data Fabric connectivity parameters are required as connector configuration parameters.

Before using Tervela Data Fabric connectors, you must configure the following items on the Tervela TPM Provisioning and Management System:

- A client user name and password to match the connector's tvauserid and tvapassword configuration parameters
- The inbound and outbound topic strings and associated schema
- Publish or subscribe entitlement rights associated with a client user name

When the connector starts, it publishes a message to topic SAS.META.tvaclientname (where tvaclientname is a connector configuration parameter). This message contains the following information:

- The mapping of the Edge Streaming Analytics engine name to a host:port field potentially used by an ESA publish/subscribe client. The host:port string is the required urlhostport connector configuration parameter, and is substituted by the engine name in topic strings used on the fabric.
- The project, query, and window names of the window associated with the connector, as well as the serialized schema of the window.

All messaging performed by the Tervela connector uses the Tervela Guaranteed Delivery mode. Messages are persisted to a Tervela TPE appliance. When a publisher connector connects to the fabric, it receives messages already published to the subscribed topic over a recent time period. By default, the publisher connector sets this time period to eight hours. This enables a publisher to catch up with a day's worth of messages. Using this mode requires regular purging of persisted data by an administrator when there are no other automated mechanism to age out persisted messages.

Tervela subscriber connectors support a hot failover mode. The active/standby status of the connector is coordinated with the fabric so that a standby connector becomes active when the active connector fails. Several conditions must be met to guarantee successful switchovers:

- The engine names of the edge streaming engines running the involved connectors must all be identical. This set of edge streaming engines is called the failover group.
- All involved connectors must be active on the same set of topics.
- All involved subscriber connectors must be configured with the same tvaclientname.

- All involved connectors must initiate message flow at the same time, and with the TPE purged of all messages on related topics. This is required because message IDs must be synchronized across all connectors.
- Message IDs that are set by the injector of event blocks into the model must be sequential and synchronized with IDs used by other standby connectors. When the injector is a Tervela publisher connector, that connector sets the message ID on all injected event blocks, beginning with ID = 1.

When a new subscriber connector becomes active, outbound message flow remains synchronized due to buffering of messages by standby connectors and coordination of the resumed flow with the fabric. The size of this message buffer is a required parameter for subscriber connectors.

Tervela connector configuration parameters named tva… are passed unmodified to the Tervela API by the connector. See your Tervela documentation for more information about these parameters.

If required, you can configure the tvapassword parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. If you have installed the Edge Streaming Analytics System Encryption and Authentication Overlay, you can use the included OpenSSL executable. Use the following command on the console to invoke OpenSSL to display your encrypted password:

```
echo "tvapassword" | openssl enc -e -aes-256-cbc -a -salt -pass
pass:"espTVAconnectorUsedByUser=tvauserid"
```

Then copy the encrypted password into your tvapassword parameter and enable the tvapasswordencrypted parameter.

Use the following parameters with Tervela connectors.

Table 8-7 Required Parameters for Subscriber Tervela Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`tvauserid`	Specifies a user name defined in the Tervela TPM. Publish-topic entitlement rights must be associated with this user name.
`tvapassword`	Specifies the password associated with tvauserid.
`tvaprimarytmx`	Specifies the host name or IP address of the primary TMX.
`tvatopic`	Specifies the topic name for the topic to which to subscribed. This topic must be configured on the TPM for the GD service and tvauserid must be assigned the Guaranteed Delivery subscribe rights for this Topic in the TPM.
`tvaclientname`	Specifies the client name associated with the Tervela Guaranteed Delivery context. If hot failover is enabled, this name must match the tvaclientname of other subscriber connectors in the failover group. Otherwise, the name must be unique among all instances of Tervela connectors.
`tvamaxoutstand`	Specifies the maximum number of unacknowledged messages that can be published to the Tervela fabric (effectively the size of the publication cache). Should be twice the expected transmit rate.

Parameter	Description
`numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector. When exceeded, the oldest message is discarded. If the connector goes active the buffer is flushed, and buffered messages are sent to the fabric as required to maintain message ID sequence.
`urlhostport`	Specifies the host/port string sent in the metadata message published by the connector on topic SAS.META.tvaclient-name when it starts.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 8-8 Required Parameters for Publisher Tervela Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`tvauserid`	Specifies a user name defined in the Tervela TPM. Subscribe-topic entitlement rights must be associated with this user name.
`tvapassword`	Specifies the password associated with tvauserid.
`tvaprimarytmx`	Specifies the host name or IP address of the primary TMX.
`tvatopic`	Specifies the topic name for the topic to which to publish. This topic must be configured on the TPM for the GD service.
`tvaclientname`	Specifies the client name associated with the Tervela Guaranteed Delivery context. Must be unique among all instances of Tervela connectors.
`tvasubname`	Specifies the name assigned to the Guaranteed Delivery subscription being created. The combination of this name and tvaclientname are used by the fabric to replay the last subscription state. If a subscription state is found, it is used to resume the subscription from its previous state. If not, the subscription is started new, starting with a replay of messages received in the past eight hours.
`urlhostport`	Specifies the host:port string sent in the metadata message published by the connector on topic SAS.META.*tvaclient-name* when it starts.

Table 8-9 Optional Parameters for Subscriber Tervela Connectors

Parameter	Description
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`hotfailover`	Enables hot failover mode.
`tvasecondarytmx`	Specifies the host name or IP address of the secondary TMX. Required if logging in to a fault-tolerant pair.

Parameter	Description
`tvalogfile`	Causes the connector to log to the specified file instead of to syslog (on Linux or Solaris) or Tervela.log (on Windows).
`tvapubbwlimit`	Specifies the maximum bandwidth, in Mbps, of data published to the fabric. The default is 100 Mbps.
`tvapubrate`	Specifies the rate at which data messages are published to the fabric, in Kbps. The default is 30,000 messages per second.
`tvapubmsgexp`	Specifies the maximum amount of time, in seconds, that published messages are kept in the cache in the Tervela API. This cache is used as part of the channel egress reliability window (if retransmission is required). The default value is 1 second.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa\EdgeStreamingServer\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`protofile`	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message.
`json`	Enables transport of event blocks encoded as JSON messages.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`tvapasswordencrypted`	Specifies that tvapassword is encrypted.

Table 8-10 Optional Parameters for Publisher Tervela Connectors

Parameter	Description
`tvasecondarytmx`	Specifies the host name or IP address of the secondary TMX. Required when logging in to a fault-tolerant pair.
`tvalogfile`	Causes the connector to log to the specified file instead of to syslog (on Linux or Solaris) or Tervela.log (on Windows)

Parameter	Description
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`protofile`	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message.
`json`	Enables transport of event blocks encoded as JSON messages.
`publishwithupsert`	Specifies to build events with opcode = Upsert instead of opcode = Insert.
`dateformat`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`tvapasswordencrypted`	Specifies that tvapassword is encrypted.
`maxevents`	Specifies the maximum number of events to publish.

8.3.2 Using the Tervela Data Fabric Adapter

Overview

The Tervela adapter supports publish and subscribe operations on a hardware-based or software-based Tervela fabric. You must install the Tervela run-time libraries to use the adapter.

dfesp_tva_adapter -C key1=val1,key2=val2,key3=val3,…

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Subscriber Usage

Table 8-11 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false. Append?collapse=true | false or ?rmretdel=true | false or both if needed.
`tvauserid=username`	Specifies the Tervela user name.
`tvapassword=password`	Specifies the Tervela password.
`tvaprimarytmx=tmx`	Specifies the Tervela primary TMX. The TMX-500 Message Switch is the primary appliance used to communicate across the Tervela data fabric.
`tvatopic=topic`	Specifies the Tervela topic.
`tvaclientname=name`	Specifies the Tervela client name.
`urlhostport=string`	Specifies the *host*:*port* string sent in connector metadata message
`tvamaxoutstand=number`	Specifies the Tervela maximum number of unacknowledged messages.
`numbufferedmsgs=number`	Specifies the maximum number of messages buffered by a standby subscriber connector.

Table 8-12 Optional Keys

Key-Value Pair	Description
`tvasecondarytmx=tmx`	Specifies the Tervela secondary TMX.
`tvalogfile=file`	Specifies the Tervela log file. The default is "syslog".
`protofile=file`	Specifies the .proto file to be used for Google protocol buffer support.
`protomsg=message`	Specifies the message itself in the .proto file that is specified by the protofile parameter.
`json=true | false`	When true, transport JSON messages instead of event blocks.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C+ + `C_dfESPpubsubSetPubsubLib()` API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.

Key-Value Pair	Description
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData\Mdsp\esa\EdgeStreamingServer\default \connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`tvapasswordencrypted=true | false`	When true, tvapassword is encrypted
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`tvapubbwlimit=bandwidth`	Specifies the Tervela maximum bandwidth of published data (Mbps). The default value is 100.
`tvapubrate=rate`	Specifies the Tervela publish rate (Kbps). The default value is 30.
`tvapubmsgexp=seconds`	Specifies the Tervela maximum time to cache published messages (seconds); the default value is 1.

Publisher Usage

Table 8-13 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/ project/continuousquery/window**.
`tvauserid=username`	Specifies the Tervela user name.
`tvapassword=password`	Specifies the Tervela password.
`tvaprimarytmx=tmx`	Specifies the Tervela primary TMX. The TMX-500 Message Switch is the primary appliance used to communicate across the Tervela data fabric.
`tvatopic=topic`	Specifies the Tervela topic.
`tvaclientname=name`	Specifies the Tervela client name.

Key-Value Pair	Description
`urlhostport=string`	Specifies the *host*:*port* string sent in connector metadata message
`tvasubname=name`	Specifies the Tervela name of the Guaranteed Delivery subscription

Table 8-14 Optional Keys

Key-Value Pair	Description
`tvasecondarytmx=tmx`	Specifies the Tervela secondary TMX.
`tvalogfile=file`	Specifies the Tervela log file. The default is "syslog".
`protofile=file`	Specifies the **.proto** file to be used for Google protocol buffer support.
`protomsg=message`	Specifies the message itself in the **.proto** file that is specified by the protofile parameter.
`json=true | false`	When true, transport JSON messages instead of event blocks.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C+ + `C_dfESPpubsubSetPubsubLib()` API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`dateformat=format`	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
`tvapasswordencrypted=true | false`	When true, tvapassword is encrypted
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka (Page 84)".

8.4 PI Connector and Adapter

8.4.1 Using the PI Connector

Overview

PI Asset Framework (PI AF) is a repository for asset-centric models, hierarchies, objects, and equipment. It is used to analyze data from multiple sources, including one or more PI Data Archives and non-PI sources such as external relational databases. Together, the metadata and time series data in the repository provide a detailed description of assets.

Using the PI Connector

The PI connector supports publish and subscribe operations for a PI AF server. The model must implement a window of a fixed schema to carry values that are associated with AF attributes owned by AF elements in the AF hierarchy. The AF data reference can be defined as a PI Point for these elements.

Note

Support for the PI connector is available only on 64–bit Microsoft Windows platforms.

The PI AF Client from OSIsoft must be installed on the Microsoft Windows platform that hosts the running instance of the connector. The connector loads the OSIsoft.AFSDK.DLL public

assembly, which requires .NET 4.0 installed on the target platform. The run-time environment must define the path to OSIsoft.AFSDK.dll.

Note

To use this connector:

- Locate the reference to this connector in the connectors: excluded: section of the file esp-properties.yml.
- Set the value to false.

The PI connector also loads the esp_pi_afsdk52 DLL located in the $DFESP_HOME/bin/plugins directory. Make sure that the run-time environment also defines the path to this DLL.

The window that is associated with the connector must use the following schema:

ID*:int32,elementindex:string,element:string,attribute:string,value:variable:,timestamp:stamp,status:string

Use the following schema values.

Schema Value	Description
elementindex	Value of the connector *afelement* configuration parameter. Specify either an AF element name or an AF element template name.
element	Name of the AF element associated with the *value*.
attribute	Name of the AF attribute owned by the AF *element*.
value	Value of the AF *attribute*.
timestamp	Timestamp associated with the *value*.
status	Status associated with the *value*.

Note

The type of the value field is determined by the type of the AF attribute as defined in the AF hierarchy. The following mapping of event stream processor data type to AF attributes is supported.

The following mapping of event stream processor data type to AF attributes is supported:

Event Stream Processor Data Type	AF Attribute
ESP_DOUBLE	TypeCode::Single TypeCode::Double
ESP_INT32	TypeCode::Byte TypeCode::SByte TypeCode::Char TypeCode::Int16 TypeCode::UInt16 TypeCode::Int32 TypeCode::UInt32

Event Stream Processor Data Type	AF Attribute
ESP_INT64	TypeCode::Int64 TypeCode::UInt64
ESP_UTF8STR	TypeCode::String
ESP_TIMESTAMP	TypeCode::DateTime

If an attribute has TypeCode::Object, the connected uses the type of the underlying PI point. Valid edge streaming analytics data types to PI point mappings are as follows:

Event Stream Processor Data Type	PI Point
ESP_INT32	PIPointType::Int16 PIPointType::Int32
ESP_DOUBLE	PIPointType::Float16 PIPointType::Float32
ESP_TIMESTAMP	PIPointType::Timestamp
ESP_UTF8STR	PIPointType::String

Use the following parameters with PI connectors:

Table 8-15 Required Parameters for Subscriber PI Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "sub".
`afelement`	Specifies the AF element or element template name. Wildcards are supported.
`iselementtemplate`	Specifies whether the afelement parameter is an element template name. By default, the afelement parameter specifies an element name. Valid values are TRUE or FALSE.
`snapshot`	Specifies whether to send snapshot data. When true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 8-16 Required Parameters for Publisher PI Connectors

Parameter	Description
`type`	Specifies to publish. Must be "pub".
`afelement`	Specifies the AF element or element template name. Wildcards are supported.
`iselementtemplate`	Specifies that the afelement parameter is an element template name. By default, the afelement parameter specifies an element name.

Table 8-17 Optional Parameters for Subscriber PI Connectors

Parameter	Description
`rmretdel`	Removes all delete events from event blocks received by the subscriber that were introduced by a window retention policy.
`pisystem`	Specifies the PI system. The default is the PI system that is configured in the PI AF client.
`afdatabase`	Specifies the AF database. The default is the AF database that is configured in the PI AF client.
`afrootelement`	Specifies the root element in the AF hierarchy from which to search for parameter afelement. The default is the top-level element in the AF database.
`afattribute`	Specifies a specific attribute in the element. The default is all attributes in the element.
`configfilesection`	Specifies the name of the section in /opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config (Linux) or %ProgramData%\Mdsp\esa\EdgeStreamingServer\default \connectors.config (Windows) to parse for configuration parameters. Specify the value as [configfilesection].

Table 8-18 Optional Parameters for Publisher PI Connectors

Parameter	Description
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`pisystem`	Specifies the PI system. The default is the PI system that is configured in the PI AF client.
`afdatabase`	Specifies the AF database. The default is the AF database that is configured in the PI AF client.
`afrootelement`	Specifies the root element in the AF hierarchy from which to search for the parameter afelement. The default is the top-level element in the AF database.
`afattribute`	Specifies a specific attribute in the element. The default is all attributes in the element.
`archivetimestamp`	Specifies that all archived values from the specified timestamp onwards are to be published when connecting to the PI system. The default is to publish only new values.
`configfilesection`	Specifies the name of the section in **/opt/mdsp/esa/config/etc/ EdgeStreamingServer/default/ connectors.config** (Linux) or **%ProgramData%\Mdsp\esa** **\EdgeStreamingServer\default** **\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`publishwithupsert`	Builds events with opcode=Upsert instead of Insert.
`maxevents`	Specifies the maximum number of events to publish.
`allvaluestostrings`	Specifies to convert all received values to a string. Requires that the valuefield in the Source window have type = ESP_UTF8STR.

8.4.2 Using the PI Adapter

Overview

The PI adapter supports publish and subscribe operations against a PI Asset Framework (PI) server. You must install the PI AF Client from OSISoft in order to use the adapter.

Note

Support for the PI adapter is available only on 64–bit Microsoft Windows platforms.

dfesp_pi_adapter -C *key1=val1,key2=val2,key3=val3,…*

Note

If a value string includes a comma, then you must enclose the entire string in escaped double quotes (for example, key=\"str,ing\").

Subscriber Usage

Table 8-19 Required Keys

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=pubsubURL`	Specifies the standard URL in the form **dfESP://host:port/project/continuousquery/window**. Append ?snapshot=true |false for subscribers. Append?collapse=true | false or ?rmretdel=true | false or both for subscribers if needed.
`afelement=element`	Specifies the AF element or element template name. Wild-cards are supported.

Table 8-20 Optional Keys

Key-Value Pair	Description
`iselementtemplate=true | false`	When true, specifies that the value of afelement is the name of an element template.
`pisystem=name`	Specifies the name of the PI system.
`afdatabase=name`	Specifies the name of the Asset Framework database.
`afrootelement=element`	Specifies a root element in the Asset Framework hierarchy from which to search for afelement.

Table 8-21 Optional Keys

Key-Value Pair	Description
`iselementtemplate=true | false`	When true, specifies that the value of afelement is the name of an element template.
`pisystem=name`	Specifies the name of the PI system.
`afdatabase=name`	Specifies the name of the Asset Framework database.
`afrootelement=element`	Specifies a root element in the Asset Framework hierarchy from which to search for afelement.
`afattribute=attribute`	Specifies an attribute in afelement and ignores other attributes there.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For tervela, the default is **./client.config**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Publisher Usage

Table 8-22 Required Keys

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=pubsubURL`	Specifies the standard URL in the form dfESP://host:port/project/continuousquery/window.
`afelement=element`	Specifies the AF element or element template name. Wildcards are supported.

Table 8-23 Optional Keys

Key-Value Pair	Description
`iselementtemplate=true | false`	When true, specifies that the value of afelement is the name of an element template.
`pisystem=name`	Specifies the name of the PI system.
`afdatabase=name`	Specifies the name of the Asset Framework database.
`afrootelement=element`	Specifies a root element in the Asset Framework hierarchy from which to search for afelement.
`afattribute=attribute`	Specifies an attribute in *afelement* and ignores other attributes there.
`gdconfig=file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
`logconfigfile=file`	Specifies the log configuration file.
`tokenlocation=location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/ subscribe server.
`configfilesection=[section]`	Specifies the name of the section in **/opt/mdsp/esa/ config/etc/ EdgeStreamingServer/ default/connectors.config** (Linux) or **%ProgramData%** **\Mdsp\esa\EdgeStreamingServer** **\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [configfilesection].
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.

Key-Value Pair	Description
`transportconfigfile=file`	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg.** For tervela, the default is **./client.config.** For rabbitmq, the default is **./rabbitmq.cfg.** For kafka, the default is **./kafka.cfg.** Note: No transport configuration file is required for native transport.
`archivetimestamp=timestamp`	Specifies that when connecting to the AF server, retrieve all archived values from the specified timestamp onwards. Timestamp is in theform "%Y-%m-%d %H:%M:%S"
`transactional=true | false`	When true, events are transactional.
`publishwithupsert=true | false`	When true, build events with opcode = Upsert instead of Insert.
`maxevents=number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window.
`allvaluestostrings=true | false`	When true, specifies to convert all received values to a string. Requires that the value field in the Source window have type = ESP_UTF8STR.

Publisher Failover

For information about implementing hot failover for publisher adapters, see “Publisher Adapter Failover with Kafka (Page 84)”.

8.5 Writing and using Custom Connectors

8.5.1 Writing and Integrating a Custom Connector

Writing a Custom Connector

When you write your own connector, the connector class must inherit from base class dfESPconnector.

Connector configuration is maintained in a set of key or value pairs where all keys and values are text strings. A connector can obtain the value of a configuration item at any time by calling getParameter() and passing the key string. An invalid request returns an empty string.

A connector can implement a subscriber that receives events generated by a window, or a publisher that injects events into a window. However, a single instance of a connector cannot publish and subscribe simultaneously.

A subscriber connector receives events by using a callback method defined in the connector class that is invoked in a thread owned by the engine. A publisher connector typically creates

a dedicated thread to read events from the source. It then injects those events into a Source window, leaving the main connector thread for subsequent calls made into the connector.

A connector must define these static data structures:

Static Data Structure	Description
`dfESPconnectorInfo`	Specifies the connector name, publish/subscribe type, initialization function pointer, and configuration data pointers.
`subRequiredConfig`	Specifies an array of dfESPconnectorParmInfo_t entries listing required configuration parameters for a subscriber.
`sizeofSubRequiredConfig`	Specifies the number of entries in subRequiredConfig.
`pubRequiredConfig`	Specifies an array of dfESPconnectorParmInfo_t entries listing required configuration parameters for a publisher.
`sizeofPubRequiredConfig`	Specifies the number of entries in pubRequiredConfig.
`subOptionalConfig`	Specifies an array of dfESPconnectorParmInfo_t entries listing optional configuration parameters for a subscriber.
`sizeofSubOptionalConfig`	Specifies the number of entries in subOptionalConfig.
`pubOptionalConfig`	Specifies an array of dfESPconnectorParmInfo_t entries listing optional configuration parameters for a publisher.
`sizeofPubOptionalConfig`	Specifies the number of entries in pubOptionalConfig.

A connector must define these static methods:

Static Method	Description
`dfESPconnector *initialize(dfESPengine *engine, dfESPpsLib_t psLib)`	Returns an instance of the connector.
`dfESPconnectorInfo *getConnectorInfo()`	Returns the dfESPconnectorInfo structure.

You can invoke these static methods before you create an instance of the connector.

A connector must define these virtual methods:

Virtual Method	Description
`start()`	Starts the connector. Must call base class method checkConfig() to validate connector configuration before starting. Must also call base class method start(). Must set variable _started = true upon success.
`stop()`	Stops the connector. Must call base class method stop(). Must leave the connector in a state whereby start() can be subsequently called to restart the connector.
`callbackFunction()`	Specifies the method invoked by the engine to pass event blocks generated by the window to which it is connected.
`errorCallbackFunction()`	Specifies the method invoked by the engine to report errors detected by the engine. Must call user callback function _errorCallback, if nonzero.
`setupCallbackFunction()`	Specifies the method invoked by the engine to set up any connector that requires the Source window schema.

A connector must set its running state for use by the connector orchestrator. It does this by calling the dfESPconnector::setState() method. The two relevant states are state_RUNNING and state_FINISHED. All connectors must set state_RUNNING when they start. Only

connectors that actually finish transferring data need to set state_FINISHED. Typically, setting state in this way is relevant only for publisher connectors that publish a finite number of event blocks.

Finally, a derived connector can implement up to ten user-defined methods that can be called from an application. Because connectors are plug-ins loaded at run time, a user application cannot directly invoke class methods. It is not linked against the connector.

The base connector class defines virtual methods userFunction_01 through userFunction_10, and a derived connector then implements those methods as needed. For example:

```
void * myConnector::userFunction_01(void *myData) {
```

An application would invoke the method as follows:

```
myRC = myConnector->userFunction_01((void *)myData);
```

Integrating a Custom Connector

All connectors are managed by a global connector manager. The default connectors shipped with Edge Streaming Analytics are automatically loaded by the connector manager during product initialization. Custom connectors built as libraries and placed in $DFESP_HOME/lib/plugins are also loaded during initialization, with the exception of those listed as excluded in esp-properties.yml.

After initialization, the connector is available for use by any event stream processor window defined in an application. As with any connector, an instance of it can be obtained by calling the window getConnector() method and passing its user-defined method. You can configure the connector using setParameter() before starting the project.

8.5.2 Using Connectors in a C++ Application

Obtaining Connectors

To obtain a new instance of a connector in a C++ application, call the dfESPwindow::getConnector() method. Pass the connector type as the first parameter, and pass a connector instance name and an "active" Boolean value as optional second and third parameters:

```
dfESPconnector *inputConn =

static_cast<dfESPconnector  *>(input-
>getConnector("fs","inputConn",  true));
```

The packaged connector types are as follows:

- adapter
- bacnet (Linux only)
- db
- fs
- kafka

- modbus
- mq
- mqtt
- nurego
- opcua
- pylon
- project
- rmq
- sniffer
- smtp
- sol
- tdata
- tdlistener
- tibrv
- timer
- tva
- url
- uvc
- websocket
- pi (Windows only)

After a connector instance is obtained, any of its base class public methods can be called. This includes setParameter(), which can be called multiple times to set required and optional parameters. Parameters must be set before the connector is started.

The type parameter is required and is common to all connectors. It must be set to pub or sub.

Additional connector configuration parameters are required depending on the connector type, and are described later in this section.

Setting Configuration Parameters

Use the setParameter() method to set required and optional parameters for a connector. You can use setParameter() as many times as you need. You must set a connector's parameters before starting it.

The type parameter is required and is common to all connectors. It must be set to pub or sub. What additional connector configuration parameters are required depends on the connector type.

Setting Configuration Parameters in a File

You can completely or partially set configuration parameters in a configuration file. You specify a set of parameters and give that set a section label. You then can use setParameter() to set the configfilesection parameter equal to the section label. This configures the entire set. If any parameters are redundant, a parameter value that you configure separately using setParameter() takes precedence.

When you configure a set of parameters, the connector finds the section label in the connectors section of the configuration file, which is in the configuration directory on page 147 . It then configures the parameters listed in that section.

The following lines specify a set of connector parameters to configure and labels the set TestConfig

```
[testconfig]
type=pub
host=localhost
port=33340
project=sub_project
continuousquery=subscribeServer
window=tradesWindow
fstype=binary
fsname=./sorted_trades1M_256perblock.bin
```

You can list as many parameters as you want in a section so labeled.